DTIC FILE COPY

(2)

AD-A225 124

ANNUAL REPORT

VOLUME 2

TASK 2: SEEKER EMULATOR DEVELOPMENT

REPORT NO. AR-0142-90-001

July 22, 1990

GUIDANCE, NAVIGATION AND CONTROL
DIGITAL EMULATION TECHNOLOGY LABORATORY

Contract No. DASG60-89-C-0142

Sponsored By

The United States Army Strategic Defense Command

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332 - 0540

Contract Data Requirements List Item A005

Period Covered: FY 90

Type Report: Annual

ANNUAL REPORT
VOLUME 2
TASK 2 SEEKER EMULATION DEVELOPMENT

July 22, 1990

Authors

Andrew M. Henshaw, Stephen R. Gieseking and Roy W. Melton

COMPUTER ENGINEERING RESEARCH LABORATORY
Georgia Institute of Technology
Atlanta, Georgia 30332 - 0540

Eugene L. Sanders	Cecil O. Alford
USASDC	Georgia Tech
Contract Monitor	Project Director

DISCLAIMER

DISCLAIMER STATEMENT - The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

DISTRIBUTION CONTROL

(1) DISTRIBUTION STATEMENT - Approved for public release; distribution is unlimited.

(2) This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227 - 7013, October 1988.

1. Advanced Seeker Scene Emulator Design

As detailed in Final Report Vol ???, the Computer Engineering Research Laboratory at the Georgia Institute of Technology and BDM Corporation have developed a real-time Focal Plane Array Seeker Scene Emulator. Using real-time, positional updates, typically from the Georgia Tech Parallel Function Processor, the Simple Seeker Scene Emulator (SSSE) can combine elements of a pre-computed database to form an image that is positionally and radiometrically correct.

Real-time operation of the Seeker Scene Emulator is achieved by precomputing target and noise data for a simulation, transferring this data to the Seeker Scene Emulator, and merging the target and noise data to produce a correct image. This image can then be processed as if it were data from an actual missile seeker sub-system.

Using the experience gained from the development of the first Seeker Scene Emulator, Georgia Tech and BDM Corporation are designing an Advanced Seeker Scene Emulator (ASSE). This emulator will address areas of concern with the SSSE and provide for more sophisticated seeker simulations.

1.1. Objectives

One problem issue with the SSSE is the use of pre-computed trajectories for the kill vehicle and targets. While this constraint is valid for many types of seeker testing. in some cases this should be avoided. Our goal with the ASSE is to avoid the use of "canned" scenarios, and additionally to provide:

- Wide FOV, Closed-Loop Operation
- Increased Fidelity Nuclear Effects Modeling
- Multispectral Capability
- Complex Threat Geometry and Dynamics
- Dynamic Tailoring/Selection of Scene Scenario
- LATS Seeker Model Anchored to LETS Test Results

1.2. Design Concept

There are two key components of real-time seeker emulation: object irradiance determination and image presentation. For an Advanced Seeker Scene Emulator, new developments in both of these areas will have to be made.

1.2.1. Object irradiance determination

The Optical Signatures Code (OSC) is the standard for current irradiance determination implementations The ASSE must perform the relevant portions of this code to be accepted by the Simulation community. Unfortunately, the OSC is computationally expensive and has always

been implemented in a non-real-time manner. For a real-time system, we are presented with two obvious choices:

- Perform OSC off-line and store information in some database for real-time retrieval
- Implement real-time OSC.

After some examination, we have decided that in order to perform the OSC off-line and still avoid a "canned scenario" implementation, the database would have to be extraordinarily large. Not only does this impact the cost of the emulator, it affects the way in which the emulator could be used – major simulation changes would take an inordinate amount of time to set up. Therefore, we are attempting a real-time implementation of the OSC.

The portion of the OSC that is relevant to GBI emulations is FASTSIG, which we are attempting to parallelize. Using this code, a network of high-speed processors should be capable of performing real-time irradiance determination for multiple objects.

1.2.2. Image presentation

Figure 1.1 and 1.2 show a comparison between the image presentation methodologies of the SSSE and the ASSE. In the ASSE, the SARIM code will be run off-line to generate Multi-Spectral Point-Spread Matrix data which will be used during run-time. This approach avoids the use of the FFT and FFT^{-1} at run-time and, instead, uses simple summing to achieve a highly accurate image representation

1.3. Requirements

Requirements for the Advanced Seeker Scene Emulator are given in the following table [BDM1]. The ASSE will provide the functionality of the SSSE and extend its capabilities in many respects. Capabilities for the SSSE are included in the table for comparison.

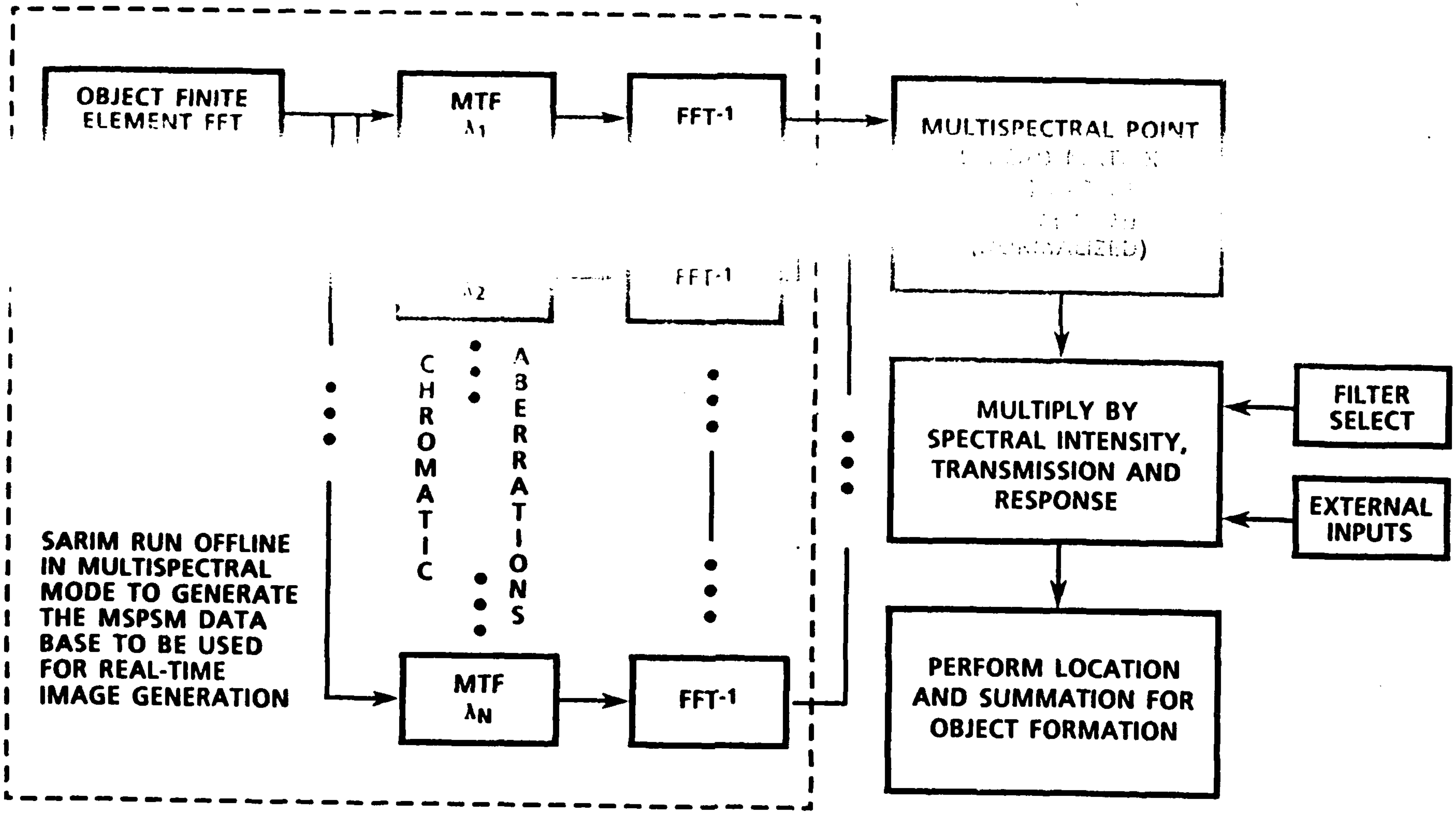
Feature	Simple Seeker Scene Emulator	Advanced Seeker Scene Emulator	Comments
Optical parameters - OTF - MTF - PSF	off-line single wavelength fft approach	off line multi-spectral mspsm approach	advanced emulator much more general accommodates multispectral time-varying characteristics
image blur	monochromatic	multispectral	accommodates n-color seeker and temperature discrimination algorithm evaluation
focal plane array detector quantum efficiency	average across spectral transmission of optics off-line calculation for non-uniformity	multispectral with general spectral response on-line calculation for non-uniformity	accommodates multiple filters and/or focal plane arrays uses mean background from previous frame accommodates time-varying signature spectral emittance
image smear	not supported	in mspsm approach	analytically based - 6 dof los rates - range closure rates - seeker look time
scenario	canned off-line real-time compensation 2D imagery	on-line generation with shadowing and crossing targets (3D) includes relative motion	provides for evaluation of algorithms addressing advanced threat characteristics
target	2-dimensional	3-dimensional	option to include high-fidelity spatial and temporal characteristics
signature	passive blackbody emittance	passive multispectral emittance multispectral illumination for diffuse and specular reflection	optical signatures code enhanced for multispectral characteristic and extended wavebands time-varying signature
transmission from target to focal plane	integrated spectral transmission of optical train	wavelength (spectral) dependent transmission of optics	provides high-fidelity diffraction and chromatic capabilities accommodates time-dependent spectral transmission (n-color)
n-color discrimination	manual input change limited to blackbody	auto-spectral switching general spectral emittance	more realistic evaluation of temperature discrimination algorithms

1.4. Schedule & Milestones

Figure 1.3 shows the schedule for development of the Advanced Seeker Scene Emulator. The milestones are:

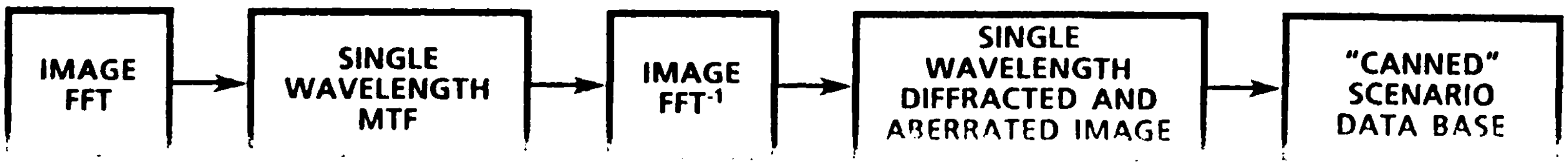
1. Delivery of feasibility study from BDM
2. Analysis of feasibility study complete.
3. Parallelization of OSC code complete.
4. Development of prototype hardware.
5. Development of software.
6. Testing of unit complete.
7. Parallel Function Processor Interface complete.
8. Interface to Georgia Tech Signal Processing Units complete.
9. Testing of entire system complete.
10. Documentation complete.

ADVANCED EMULATOR - FINITE ELEMENT PROCESSING (EMULATOR 2)



R1-9808G-C1

PRESENT EMULATOR–OBJECT IMAGE PROCESSING (EMULATOR 1)



2. Interfaces

2.1. LATS

Within the LATS/LETS system (Figure 2.1), the Adaptable Simulator Environment (ASE) will provide time-dependent processing. The object-dependent processing algorithms are to be performed in real-time by the Parallel Function Processor.

The transition from pixel to objects will occur at the interface between the PFP and the ASE. The interface (see Figure 2.2) will be Transputer-based and will convert the 32x32 pixels per frame received over a SCSI channel to an arbitrary number of objects transmitted over a 10 Mbit per second Inmos-protocol link.

2.1.1. SCSI interface

There are two SCSI channels for communication between the ASE and the PFP. One channel is dedicated to the transfer of processed pixel data from the ASE to the PFP. The other channel is provided for lower volume communications, such as response feedback, coefficient updating, etc. Both channels are differentially-driven SCSI capable of approximately 1.5 MBytes/sec data transfer. Physical and electrical specifications are as specified in ANSI X.131-1986.

Currently, the format for the data flow has only been specified for the pixel data flowing from the ASE to the PFP. The format is as follows:

Type: Time-dependent processing pixel data.

Size: 16-bit word, 1024 words/frame (32 rows x 32 columns)

Format: Sequential row-column

Rate: Maximum of approximately 205 Kbytes/sec at 100 frames/sec

In the PFP/LATS interface, the SCSI interface will be handled by a Rancho Technology RT-SDA-M differential/single-ended SCSI converter and an Inmos IMS B422 SCSI Interface Transputer module.

2.1.2. Inmos-protocol link interface

The interface from the Clustering Engine to the PFP is a simple iSBX board that carries a single Inmos Link Adaptor and some interface circuitry (Figure 2.??). An alternative to the use of this board would be the GT-XBI (Transputer Crossbar Interface) that allows direct connection to a crossbar port on the PFP. However, the crossbar support software must know *a priori* the number of transfers that will occur, and, because the number of clusters found per frame is not fixed, the crossbar communications would have to be set for some maximum value.

The interface supports the standard Inmos link protocols of single-ended 10 and 20 Mbits/sec transfers. Additionally, differential drivers can be enable to provide data transfers over longer distances than the standard one meter.

2.1.3. Clustering Engine

The Clustering Engine will not be field programmable. Georgia Tech will provide the code for the unit in ROMs which are present in the design(see below). The design has sufficient computing power to provide for other functions not currently specified, but it has been decided any functions added at some later date be provided by the Computer Engineering research Laboratory.

2.1.3.1. Hardware

As shown in Figure 2.2, the hardware for the Clustering Engine will be based upon the Inmos Transputer. Four processor modules (IMS B401-3) will perform the clustering operation while the other modules are provided for interface and program storage.

Module ID	Operation	Description
IMS-B401	Clustering processor	T414 Transputer with 32 Kbytes RAM
IMS B422	SCSI Interface	T222 Transputer with SCSI interface
IMS B418	Read-only memory for booting all processors	128Kbytes ROM
IMS B415	Differential/Standard link converter	

2.1.3.2. Software

Georgia Tech defines the clustering operation as follows:

> Clustering consists of associating a label with each non-zero valued pixel in an MxN image of pixels, with the condition that two pixels have the same label if and only if they lie in the same connected component. A connected component is defined as a maximal region of non-zero valued pixels such that any two pixels in the region lie on a connected path passing only through pixels with a non-zero value. Two conventional definitions for connectedness are 4-connectedness, adjacent vertically and horizontally, and 8-connectedness, adjacent vertically, horizontally, and diagonally._
>
> _ Paraphrased from R.E. Cypher et al., "Algorithms for Image Component Labeling on SIMD Mesh-Connected Computers," *IEEE Transactions On Computers* , Vol 39, No. 2, February 1990.

The listing for a test of the clustering algorithm is given in Appendix A. This test shows the capability of two Transputers to perform the clustering of a 32x32 pixel array at 100 Hz rates.

2.2. AEDC

Two other interfaces to the system are expected to be required - SCSI and Ethernet channels. These interfaces are designed to support control communications between the PFP and some data collection, monitoring, or control computers.

2.2.1. SCSI

The SCSI interface will be provided as a single Multibus I (or Multibus II, depending on the configuration of the PFP) board that will plug directly into the PFP. This board will take data from the crossbar and relay the relevant values to the control systems. The particular SCSI interface board has not been chosen yet.

2.2.2. Ethernet

The Ethernet interface from the PFP to the AEDC systems will simply be the standard Ethernet board supplied with the PFP host (Sun 386i, Intel 310, etc.).

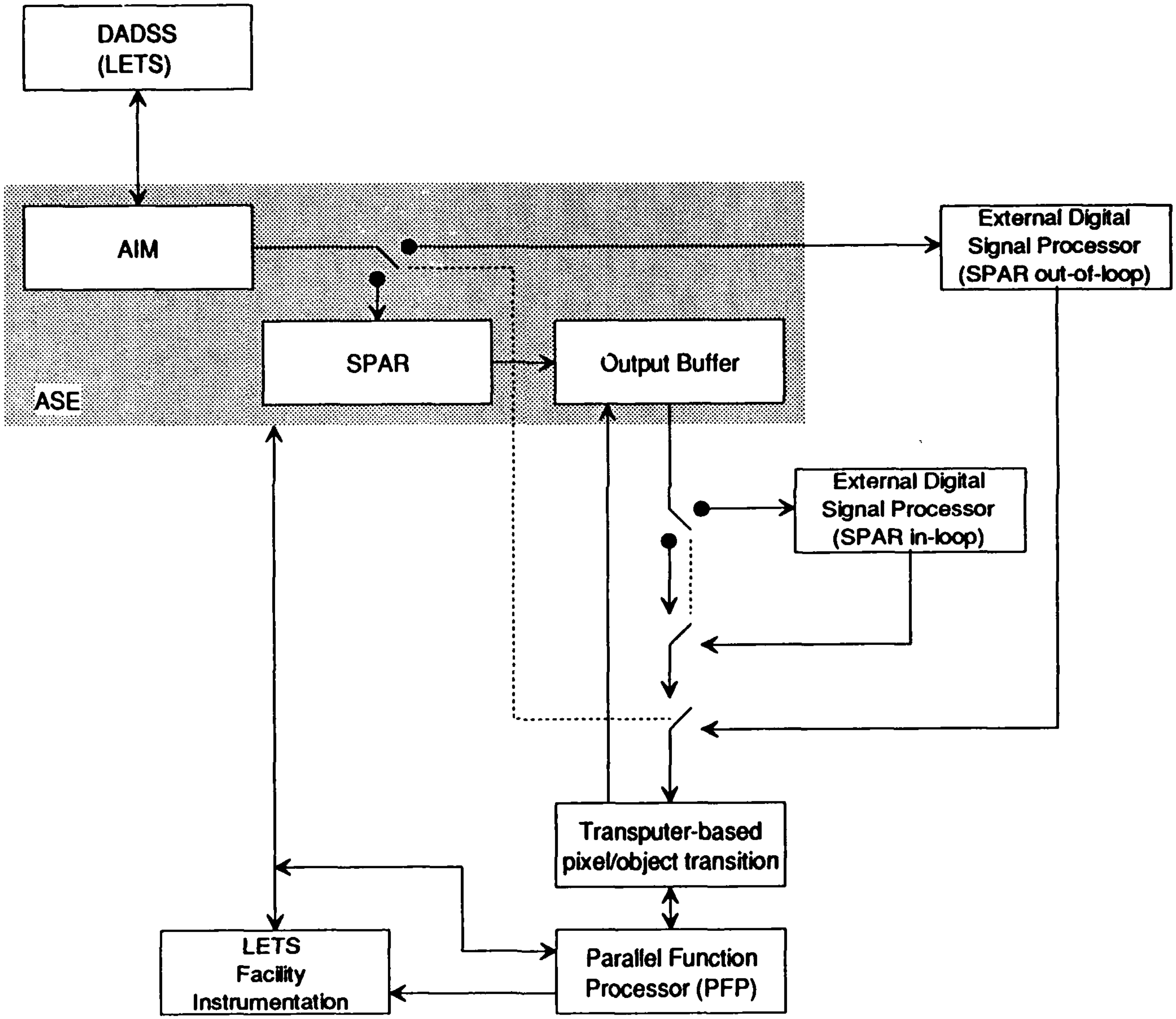
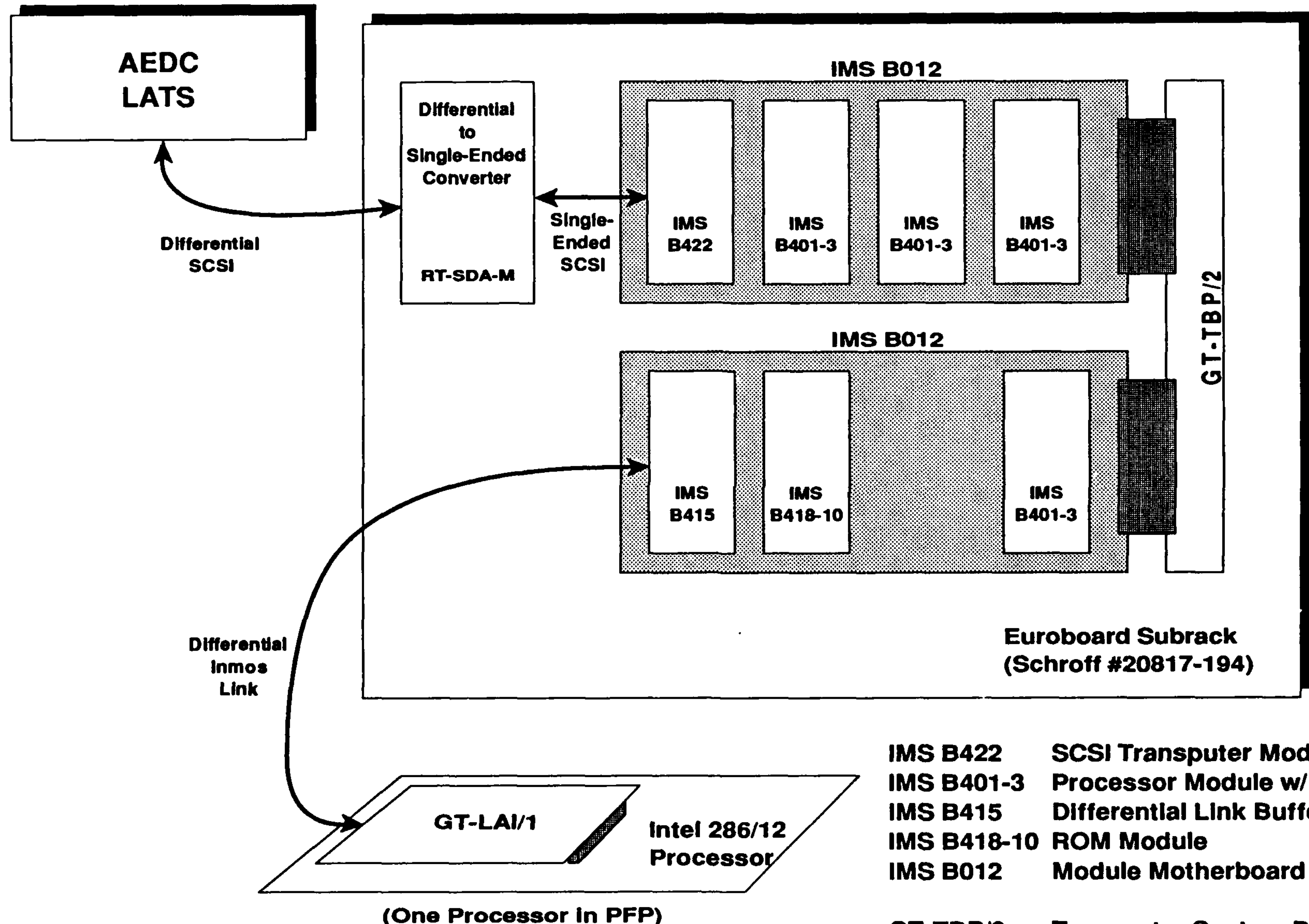

DADSS
(LETS)
AIM
SPAR
Output Buffer
ASE
External Digital
Signal Processor
(SPAR out-of-loop)
External Digital
Signal Processor
(SPAR in-loop)
Transputer-based
pixel/object transition
Parallel Function
Processor (PFP)
LETS
Facility
Instrumentation

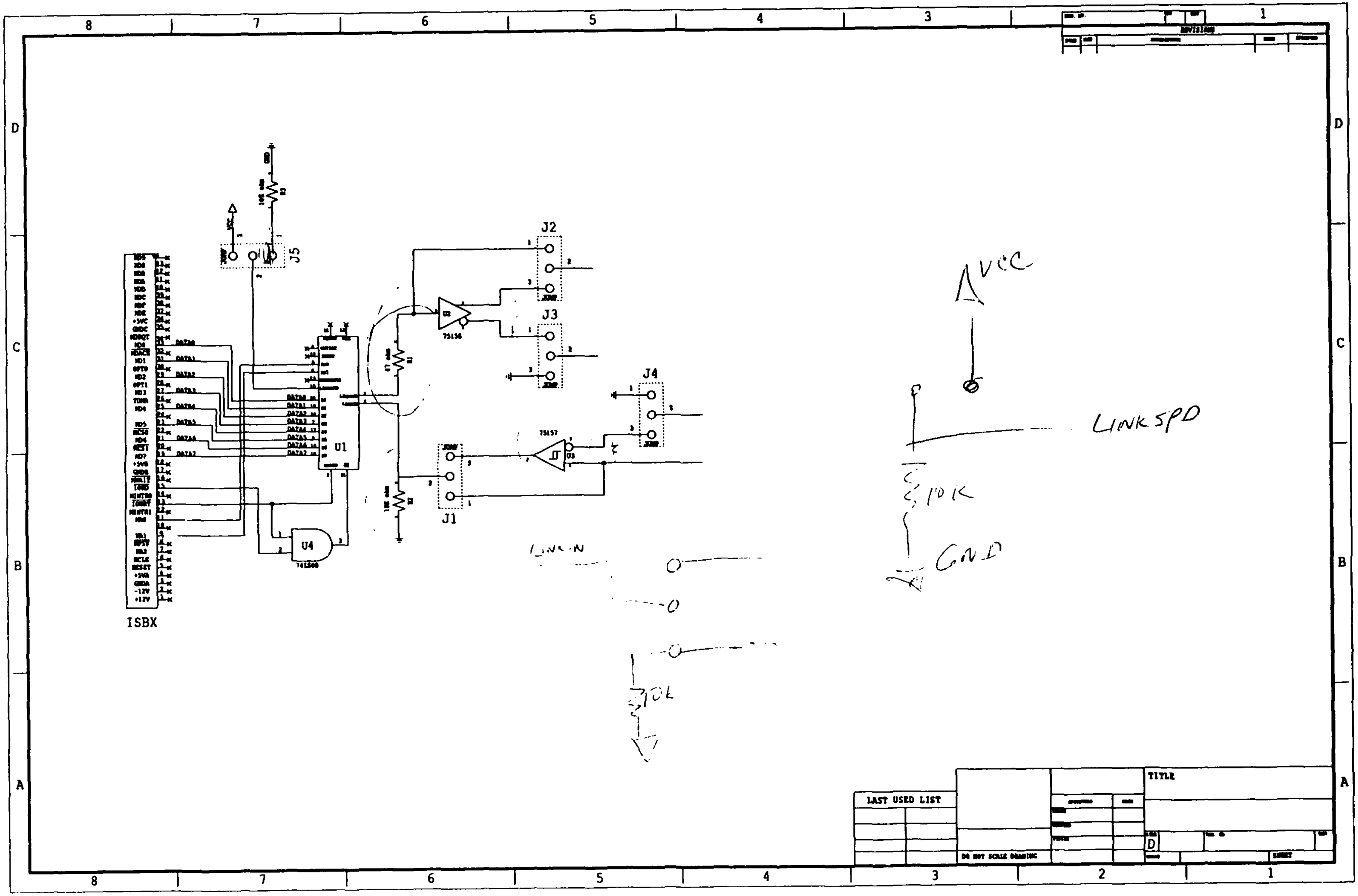
Figure 2.1



IMS B422	SCSI Transputer Module
IMS B401-3	Processor Module w/ 32K & T800
IMS B415	Differential Link Buffer Module
IMS B418-10	ROM Module
IMS B012	Module Motherboard
GT-TBP/2	Transputer System Backplane
GT-LAI/1	Link Adaptor - ISBX Module

~~Figure 1~~. PFP/LATS Interface

Figure 2.2



ISBX
J1
J2
J3
J4
J5
U1
U2
U3
U4
75150
75157
74LS08
VCC
GND
DATA0
DATA1
DATA2
DATA3
DATA4
DATA5
DATA6
DATA7
LINKSPD
10K
GND
TITLE
LAST USED LIST
DO NOT SCALE DRAWING
D

Fig 2.3

3. Signal Processing Algorithms

This section describes a set of signal-processing algorithms, as implemented by the Computer Engineering Research Laboratory at Georgia Tech. The routines are presented as a representative collection of operations for processing Infrared Focal-Plane Array signals.

For the purposes of testing and dissemination, each algorithm is presented as a stand-alone FORTRAN program. These programs are based upon a core *harness* routine which supports the input/output of a common data format (Georgia Tech Algorithm Evaluation Data Format - described in the Harness section). The modular implementations offer several benefits:

- simplification of the generation of test vectors for the verification of alternate implementations
- capability for testing various algorithm combinations, without re-compilation
- support for multiple language and/or processor-platform implementations

3.1. Harness

3.1.1. Description

The *Harness* program shown below is the basis of the input/output methodology used by all of the routines in this document. The code implements a simple Pass-Through module which reads a data stream, picking off the FPA pixel data, and writing the data onto an output data stream.

The Georgia Tech Algorithm Evaluation Data format is a simple ASCII text representation of a data stream. The data stream has two major components - the *Field Header* and the *Field Data*. The harness of each module processes the data stream by reading each line and checking for Field Headers which are relevant to that module. Any lines which are not relevant, or unrecognized, are immediately placed upon the output data stream. As soon as a relevant Field Header is recognized, the Field Data which follows is processed in a manner which is appropriate to that module and Field Header. This scheme provides for the chaining of modules output-to-input, without either module requiring knowledge of all, or any, of the other module's data formats. In typical use, controls for many modules could be included in a single data stream; each module would only process data intended for it. For example, suppose a test setup was composed of the following pipeline:

Input data stream ----> Spatial Filter ----> Simple Threshold ----> Output Data Stream

The data stream might appear as follows:

Input Data Stream	Description	Used by	Action
Dimensions 128	Field Header Field Data	Spatial Filter and Simple Threshold	input
Simple Thresholding Limits 0 256	Field Header Field Data	Simple Threshold	input
Pixel Data 99 93 ... 76	Field Header Field Data Field Data	Spatial Filter and Simple Threshold	input, modified
End	Field Header	Spatial Filter and Simple Threshold	input

Output Data Stream	Description	Generated by	Action
Dimensions 128	Field Header Field Data	Spatial Filter and Simple Threshold	copied to output data stream
Simple Thresholding Limits 0 256	Field Header Field Data	Simple Threshold	copied to output data stream
Simple Thresholding Statistics 0 256 1024	Field Header Field Data	Simple Threshold	generated, then placed on output data stream
Pixel Data 99 93 ... 76	Field Header Field Data Field Data	Spatial Filter and Simple Threshold	modified, then placed on output data stream
End	Field Header	Spatial Filter and Simple Threshold	copied to output data stream

3.1.2. Module Listing

```
      PROGRAM HARNESS
C
C     This program acts as a harness for testing various
C     Signal Processing Routines
C
```

```
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
C     400 Tenth St.  CRB 390
C     Atlanta, GA  30332-0540
C     Contact: Andrew Henshaw  (404)894-2521
C
C     Written by A. M. Henshaw    Jan 23, 1990
C     Using Microsoft Fortran
C
      CHARACTER*(*) Dim, Pixels
      PARAMETER (Dim   ='Dimensions')
      PARAMETER (Pixels='Pixel Data')
      PARAMETER (maxSize=64)

      INTEGER n
      INTEGER in(maxSize,maxSize), out(maxSize, maxSize)
      CHARACTER header*72
      LOGICAL runFlag

      WRITE (6,*) '% Processed by Pass Thru module.'
      runFlag = .TRUE.
      DO WHILE (runFlag)
        READ (5,1000) header
1000    FORMAT (A72)
        IF (header.EQ.Dim) THEN
          READ (5,*) n
          WRITE (6,*) Dim
          WRITE (6,*) n
        ELSE IF (header.EQ.Pixels) THEN
          READ (5,*) ((in(row,col),col=1,n),row=1,n)

          CALL PassThru (n, in, out)

          WRITE (6,*) Pixels
          WRITE (6,*) ((out(row,col),col=1,n),row=1,n)
        ELSE IF (header.EQ.'End') THEN
          WRITE (6,*) 'End'
          runFlag = .FALSE.
        ELSE
          WRITE (6,*) header
        END IF
      END DO

      END
```

```
C*********************************************************************

      SUBROUTINE PassThru (n, in, out)

      PARAMETER (maxSize=64)
      INTEGER n, row, col
      INTEGER in(maxSize, maxSize)
      INTEGER out(maxSize, maxSize)


      DO 30 row = 1, n
        DO 30 col = 1, n
          out(row,col) = in(row,col)
30    CONTINUE

      RETURN
      END
```

3.2. Non-Uniformity Compensation

3.2.1. Description

The non-uniformity compensation algorithm provides a pixel-by-pixel correction of the actual pixel response to the desired response. The current algorithm uses up to a five-point, piecewise-linear correction to the pixel intensity. The correction is determined by sending a specified number of calibration frames through the process. Each of the calibration frames will have been generated by exposing the focal-plane array to a known intensity so that a desired pixel intensity is expected at each pixel.

Dead, or inadequately responsive, pixels are assumed to have been marked by another module and, during processing, they are replaced by the intensity of the previous pixel.

After calibration is performed, the algorithm enters the processing phase. For each pixel which is to be processed, it is first determined if it is a dead pixel. For normal pixels, the calibration intensities are searched to determine which section should be used for correction. After the section is determined, a linear interpolation is performed using the input pixel intensity to interpolate between the desired responses.

3.2.2. Data fields

Action	Field Header	Field Data	Data Type
input	*Dimensions*	FPA dimension	Integer
input	*Calibration Frames*	Count of calibration frames	Integer
input	*Calibration Input*	Vector of reference inputs	Integer [1..Count]
input	*Calibration Pixel Data*	Array of pixel response data for one input reference	Integer [1..Dimension] [1..Dimension]
modify	*Pixel Data*	Pixel data array	Integer [1..Dimension] [1..Dimension]

3.2.3. Module Listing

```
      PROGRAM NUNICOMP
C
C     Non-Uniform Compensation Test Module
C
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
C     400 Tenth St.  CRB 390
C     Atlanta, GA  30332-0540
C     Contact: Andrew Henshaw  (404)894-2521
C
C     conforms to the Ga. Tech Algorithm Evaluation Data Format
C
C     Fortran translation of Occam code
C     Steve Gieseking
C     Roy W. Melton     Feb  1, 1990
C
C     Harness written by Andrew Henshaw     Jan 23, 1990
C     Using Microsoft Fortran
C
      CHARACTER*(*) CalInp, CalOutp, Dim, Pixels, Sect
C
C     Valid Section Headers
C
      PARAMETER (CalInp  = 'Calibration Input')
      PARAMETER (CalOutp = 'Calibration Pixel Data')
      PARAMETER (Dim     = 'Dimensions')
      PARAMETER (Pixels  = 'Pixel Data')
```

```
      PARAMETER (Sect    = 'Calibration Frames')
C
      PARAMETER (maxSize = 128)      ! maximum FPA size
      PARAMETER (maxCalFrames = 5)   ! default value

      INTEGER N, Count, Sections
      INTEGER Ic (maxCalFrames)
      INTEGER In (maxSize, maxSize), Out(maxSize, maxSize)
      INTEGER Oc (maxCalFrames, maxSize, maxSize)
      CHARACTER Header*72
      LOGICAL runFlag

      Count = 1
      Sections = maxCalFrames
      WRITE (6,*) '% Processed by Non-Uniformity Compensation Module.'
      runFlag = .TRUE.
      DO WHILE (runFlag)
        READ (5,1000) Header
1000    FORMAT (A72)
        IF (Header.EQ.CalInp) THEN
          IF (Count.LE.Sections) THEN
            READ (5,*) Ic (Count)
            WRITE (6,*) CalInp
            WRITE (6,*) Ic (Count)
          ELSE
            WRITE (6,*) CalInp
          ENDIF

        ELSEIF (Header.EQ.CalOutp) THEN
          IF (Count.LE.Sections) THEN
            READ (5, *) ((Oc (Count, Row, Col), Col=1,N), Row=1,N)
            WRITE (6,*) CalOutp
            WRITE (6,*) ((Oc (Count, Row, Col), Col=1,N), Row=1,N)
            Count = Count + 1
          ELSE
            WRITE (6,*) CalOutp
          ENDIF

        ELSEIF (Header.EQ.Dim) THEN
          READ (5,*) N
          WRITE (6,*) Dim
          WRITE (6,*) N

        ELSE IF (Header.EQ.Pixels) THEN
```

```
        IF ((Count.GT.1).AND.(Sections.GT.1)) THEN
          READ (5,*) ((In(row,col),col=1,n),row=1,n)
          IF (Count.LE.Sections) THEN
            Sections = Count - 1
          ENDIF
          CALL NonUniformityCompensation
     +              (In, Out, Oc, Ic, N, Sections)
          WRITE (6,*) Pixels
          WRITE (6,*) ((Out(row,col), col=1,n), row=1,n)
        ELSE
          WRITE (6,*) Pixels
        ENDIF

      ELSEIF (Header.EQ.Sect) THEN
        READ (5,*) Sections
        WRITE (6,*) Sect
        WRITE (6,*) Sections

      ELSE IF (Header.EQ.'End') THEN
        WRITE (6,*) 'End'
        runFlag = .FALSE.
      ELSE
        WRITE (6,*) header
      END IF
    END DO

    END

C**********************************************************************

      SUBROUTINE NonUniformityCompensation
     +              (In, Out, Oc, Ic, N, Sections)

      PARAMETER (MAXCALFRAMES=5)
      PARAMETER (MAXSIZE=64)
      INTEGER In (MAXSIZE, MAXSIZE), Out (MAXSIZE, MAXSIZE)
      INTEGER Oc (MAXCALFRAMES, MAXSIZE, MAXSIZE)
      INTEGER Ic (MAXCALFRAMES)
      INTEGER N, Sections
      INTEGER I, J, LastPixel, Section


      LastPixel = 0
      DO 20 I = 1, N
```

```
      DO 20 J = 1, N
        IF (Oc (1, I, J).EQ.65535) THEN
          Out (I, J) = LastPixel
        ELSE
          Section = 1
10        IF ((Section.LT.(Sections - 1)).AND.
     +        (In (I, J).GE.Oc ((Section + 1), I, J))) THEN
            Section = Section + 1
            GOTO 10
          ENDIF

          IF (In (I, J).LT.Oc (Section, I, J)) THEN
            Out (I, J) = Oc (Section, I, J)
          ELSE
            Out (I, J) = (In (I, J) - OC (Section, I, J)) *
     +                   (Ic (Section + 1) - Ic (Section)) /
     +                   (Oc ((Section + 1), I, J) -
     +                    Oc (Section, I, J)        ) +
     +                   Ic (Section)
          ENDIF

          IF (Out (I, J).GT.65535) THEN
            Out (I, J) = 65535
          ENDIF
          LastPixel = Out (I, J)
        ENDIF
20    CONTINUE
      RETURN
      END
```

3.3. Spatial Filtering

3.3.1. Description

The spatial filtering algorithm performs a convolution of the image with a 3x3 coefficient mask. This implementation supports a four point symmetric mask. Separate masks are used for the edge pixels since not all of the pixels which are needed are defined. This allows more general application of boundary conditions than would be available if the undefined pixels were treated as zeros and the same mask was used.

Since the filter allows negative coefficients in the mask, it is possible to generate negative output intensities. The coding allows the intensity to be output limited to a positive range.

3.3.2. Data Fields

Action	Field Header	Field Data	Data Type
input	*Dimensions*	FPA dimension	Integer
input	*Spatial Filter Controls*	Filter coefficients (Corner coefficients)	Integer [1..4]
		Filter coefficients (Top coefficients)	Integer [1..4]
		Filter coefficients (Right coefficients)	Integer [1..4]
		Filter coefficients (Center coefficients)	Integer [1..4]
modify	*Pixel Data*	Pixel data array	Integer [1..Dimension] [1..Dimension]

3.3.3. Module Listing

```
      PROGRAM SPFILT
C
C     Spatial Filtering Test Module
C
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
C     400 Tenth St.  CRB 390
C     Atlanta, GA  30332-0540
C     Contact: Andrew Henshaw  (404)894-2521
C
C     conforms to the Ga. Tech Algorithm Evaluation Data Format
C
C     Fortran translation of Occam code
C     Steve Gieseking
C     Roy Melton
C
C     Harness written by Andrew Henshaw     Jan 23, 1990
C     Using Microsoft Fortran
C
      CHARACTER*(*) Controls, Dim, Pixels
      PARAMETER (Controls = 'Spatial Filter Controls')
      PARAMETER (Dim      = 'Dimensions')
      PARAMETER (Pixels   = 'Pixel Data')
      PARAMETER (maxSize  = 64)
```

```
      PARAMETER (SF_CONTROL_SIZE = 4)

      INTEGER N
      INTEGER In (maxSize, maxSize), Out (maxSize, maxSize)
      INTEGER C (SF_CONTROL_SIZE, SF_CONTROL_SIZE)

      CHARACTER header*72
      LOGICAL runFlag

      WRITE (6,*) '% Processed by Spatial Filtering Module.'

      CALL DefaultFilterControls (C)
      runFlag = .TRUE.

      DO WHILE (runFlag)
        READ (5,1000) header
 1000   FORMAT (A72)
        IF (header.EQ.Controls) THEN
          READ (5,*) ((C (row, col), col=1, SF_CONTROL_SIZE),
     +                               row=1, SF_CONTROL_SIZE  )
          WRITE (6,*) Controls
          WRITE (6,*) ((C (row, col), col=1, SF_CONTROL_SIZE),
     +                                row=1, SF_CONTROL_SIZE  )
        ELSE IF (header.EQ.Dim) THEN
          READ (5,*) N
          WRITE (6,*) Dim
          WRITE (6,*) N
        ELSE IF (header.EQ.Pixels) THEN
          READ (5,*) ((In(row,col),col=1,n),row=1,n)
          CALL SpatialFilter (In, Out, C, N)
          WRITE (6,*) Pixels
          WRITE (6,*) ((Out(row,col),col=1,n),row=1,n)
        ELSE IF (header.EQ.'End') THEN
          WRITE (6,*) 'End'
          runFlag = .FALSE.
        ELSE
          WRITE (6,*) header
        END IF
      END DO

      END

C***Filter Control Defaults*******************************************
      SUBROUTINE DefaultFilterControls (Control)
```

```
      PARAMETER (SF_CONTROL_SIZE = 4)

      INTEGER Control (SF_CONTROL_SIZE, SF_CONTROL_SIZE)
      INTEGER I, J

      DO 210 I = 1, 4
        DO 200 J = 1, 3
          Control (I, J) = 0
  200   CONTINUE

        Control (I, 4) = 16384
  210 CONTINUE

      RETURN
      END

C***Temporal Filter*****************************************************
      SUBROUTINE SpatialFilter (In, Out, C, N)

      PARAMETER (MAXSIZE = 64)
      PARAMETER (SF_CONTROL_SIZE = 4)

      INTEGER In (MAXSIZE, MAXSIZE), Out (MAXSIZE, MAXSIZE)
      INTEGER C (SF_CONTROL_SIZE, SF_CONTROL_SIZE)
      INTEGER N
      INTEGER I, J

      DO 100 I = 1, N
        DO 100 J = 1, N
          IF (I.EQ.1) THEN
            IF (J.EQ.1) THEN
              Out (I, J) = (In (I+1, J+1) * C (1, 1)) +
     +                     (In (I, J+1)   * C (1, 2)) +
     +                     (In (I+1, J)   * C (1, 3)) +
     +                     (In (I, J)     * C (1, 4))
            ELSEIF (J.EQ.N) THEN
              Out (I, J) = (In (I+1, J-1) * C (1, 1)) +
     +                     (In (I, J-1)   * C (1, 2)) +
     +                     (In (I+1, J)   * C (1, 3)) +
     +                     (In (I, J)     * C (1, 4))
            ELSE
              Out (I, J) = ((In (I+1, J-1) + In (I+1, J+1)) * C (2, 1))+
     +                     ((In (I, J-1)   + In (I, J+1))   * C (2, 2))+
```

```
     +                 (In (I+1, J)                    * C (2, 3))+
     +                 (In (I, J)                      * C (2, 4))
          ENDIF
        ELSEIF (I.EQ.N) THEN
          IF (J.EQ.1) THEN
            Out (I, J) = (In (I-1, J+1) * C (1, 1)) +
     +                   (In (I, J+1)   * C (1, 2)) +
     +                   (In (I-1, J)   * C (1, 3)) +
     +                   (In (I, J)     * C (1, 4))
          ELSEIF (J.EQ.N) THEN
            Out (I, J) = (In (I-1, J-1) * C (1, 1)) +
     +                   (In (I, J-1)   * C (1, 2)) +
     +                   (In (I-1, J)   * C (1, 3)) +
     +                   (In (I, J)     * C (1, 4))
          ELSE
            Out (I, J) = ((In (I-1, J-1) + In (I-1, J+1)) * C (2, 1))+
     +                   ((In (I, J-1)   + In (I, J+1))   * C (2, 2))+
     +                   (In (I-1, J)                     * C (2, 3))+
     +                   (In (I, J)                       * C (2, 4))
          ENDIF
        ELSEIF (J.EQ.1) THEN
          Out (I, J) = ((In (I-1, J+1) + In (I+1, J+1)) * C (3, 1)) +
     +                 (In (I, J+1)                     * C (3, 2)) +
     +                 ((In (I-1, J)   + In (I +1, J))  * C (3, 3)) +
     +                 (In (I, J)                       * C (3, 4))
        ELSEIF (J.EQ.N) THEN
          Out (I, J) = ((In (I-1, J-1) + In (I+1, J-1)) * C (3, 1)) +
     +                 (In (I, J-1)                     * C (3, 2)) +
     +                 ((In (I-1, J) + In (I+1, J))     * C (3, 3)) +
     +                 (In (I, J)                       * C (3, 4))
        ELSE
          Out (I, J) = ((In (I-1, J-1) + In (I+1, J-1) +
     +                   In (I-1, J+1) + In (I+1, J+1) ) * C (4, 1)) +
     +                 ((In (I, J-1)   + In (I, J+1))    * C (4, 2)) +
     +                 ((In (I-1, J)   + In (I+1, J))    * C (4, 3)) +
     +                 (In (I, J)                        * C (4, 4))
        ENDIF

        Out (I, J) = Out (I, J) / 16384
        IF (Out (I, J).LT.0) THEN
          Out (I, J) = 0
        ELSEIF (Out (I, J).GT.65535) THEN
          Out (I, J) = 65535
        ENDIF
```

```
  100 CONTINUE

      RETURN
      END
```

3.4. Temporal Filtering

3.4.1. Description

The temporal filtering algorithm provides a pixel-by-pixel infinite impulse response (IIR) filtering of the sequence of images which are sent through the process. This implementation allows up to a fourth-order filter.

3.4.2. Data Fields

Action	Field Header	Field Data	Data Type
input	*Dimensions*	FPA dimension	Integer
input	*Temporal Filtering Limits*	Lower limits	Integer
		Upper Limits	Integer
modify	*Pixel Data*	Pixel data array	Integer [1..Dimension] [1..Dimension]

3.4.3. Module Listing

```
      PROGRAM TEMPORALFILTER
C
C     Temporal Filter Test Module
C
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
C     400 Tenth St.  CRB 390
C     Atlanta, GA  30332-0540
C     Contact: Andrew Henshaw  (404)894-2521
C
C     conforms to the Ga. Tech Algorithm Evaluation Data Format
C
C     Fortran translation of Occam code
C     Steve Gieseking
C     Roy Melton
C
```

```
C     Harness written by A. M. Henshaw     Jan 23, 1990
C     Using Microsoft Fortran
C
      CHARACTER*(*) Dim, Pixels, Limits
C
C     Valid Section Headers
      PARAMETER (Dim   ='Dimensions')
      PARAMETER (Pixels='Pixel Data')
      PARAMETER (Limits='Temporal Filtering Limits')
C
      PARAMETER (maxSize=64)      ! maximum FPA size
      PARAMETER (TF_CONTROL_SIZE = 24)

      INTEGER n, lower, upper
      INTEGER in(maxSize,maxSize), out(maxSize, maxSize)
      INTEGER X (maxSize, maxSize, 2, 2)
      INTEGER C (TF_CONTROL_SIZE)

      CHARACTER header*72
      LOGICAL runFlag
      DATA lower /0/
      DATA upper /32767/

      WRITE (6,*) '% Processed by Temporal Filtering Module.'
      runFlag = .TRUE.
      DO WHILE (runFlag)
        READ (5,1000) header
 1000   FORMAT (A72)
        IF (header.EQ.Dim) THEN
          READ (5,*) n
          WRITE (6,*) Dim
          WRITE (6,*) n
        ELSE IF (header.EQ.Limits) THEN
          READ (5,*) lower, upper
          WRITE (6,*) Limits
          WRITE (6,*) lower, upper
        ELSE IF (header.EQ.Pixels) THEN
          READ (5,*) ((in(row,col),col=1,n),row=1,n)

          CALL CalculateFilterControls (C, Lower, Upper)

          CALL TempFilt (In, Out, X, C, N)

          WRITE (6,*) Pixels
```

```
          WRITE (6,*) ((out(row,col),col=1,n),row=1,n)
        ELSE IF (header.EQ.'End') THEN
          WRITE (6,*) 'End'
          runFlag = .FALSE.
        ELSE
          WRITE (6,*) header
        END IF
      END DO

      END

C***Filter Control Calculation****************************************
      SUBROUTINE CalculateFilterControls (Control, Lower, Upper)

      PARAMETER (TF_CONTROL_SIZE = 24)

      INTEGER Control (TF_CONTROL_SIZE)
      INTEGER Lower, Upper
      INTEGER I, J

      DO 110 I = 0, 12, 12
        DO 100 J = 1, 8
          Control (J + I) = 1
  100   CONTINUE

        Control (9 + I) = 3
        Control (10 + I) = 1
        Control (11 + I) = Upper
        Control (12 + I) = Lower
  110 CONTINUE

      RETURN
      END

C***Temporal Filter***************************************************
      SUBROUTINE TempFilt (In, Out, X, C, N)

      PARAMETER (MAXSIZE = 64)
      PARAMETER (TF_A0 = 1)
      PARAMETER (TF_A1 = 2)
      PARAMETER (TF_A2 = 3)
      PARAMETER (TF_B0 = 4)
      PARAMETER (TF_B1 = 5)
      PARAMETER (TF_B2 = 6)
```

```
      PARAMETER (TF_SCALE_STATE = 7)
      PARAMETER (TF_SCALE_OUTPUT = 8)
      PARAMETER (TF_UPPER_LIMIT_STATE = 9)
      PARAMETER (TF_LOWER_LIMIT_STATE = 10)
      PARAMETER (TF_UPPER_LIMIT_OUTPUT = 11)
      PARAMETER (TF_LOWER_LIMIT_OUTPUT = 12)
      PARAMETER (TF_CONTROL_SIZE = 24)

      INTEGER In (MAXSIZE, MAXSIZE), Out (MAXSIZE, MAXSIZE)
      INTEGER X (MAXSIZE, MAXSIZE, 2, 2)
      INTEGER C (TF_CONTROL_SIZE)
      INTEGER N
      INTEGER I, J, K, L, Ptr, Value, XNew, YNew

      DO 10 I = 1, MAXSIZE
        DO 10 J = 1, MAXSIZE
          DO 10 K = 1, 2
            DO 10 L = 1, 2
              X (I, J, K, L) = 0
   10 CONTINUE

      DO 30 I = 1, N
        DO 30 J = 1, N
          Value = In (I, J)
          DO 20 K = 1,2
            Ptr = (K - 1) * 12
            XNew = ((C (Ptr + TF_A0) * Value) +
     +               (C (Ptr + TF_A1) * X (I, J, K, 1)) +
     +               (C (Ptr + TF_A2) * X (I, J, K, 2))    ) /
     +              C (Ptr + TF_SCALE_STATE)
            YNew = ((C (Ptr + TF_B0) * Value) +
     +               (C (Ptr + TF_B1) * X (I, J, K, 1)) +
     +               (C (Ptr + TF_B2) * X (I, J, K, 2))    ) /
     +              C (Ptr + TF_SCALE_OUTPUT)

            X (I, J, K, 2) = X (I, J, K, 1)

            IF (XNew.GT.C (Ptr + TF_UPPER_LIMIT_STATE)) THEN
              X (I, J, K, 1) = C (Ptr + TF_UPPER_LIMIT_STATE)
            ELSEIF (XNew.LT.C (Ptr + TF_LOWER_LIMIT_STATE)) THEN
              X (I, J, K, 1) = C (Ptr + TF_LOWER_LIMIT_STATE)
            ELSE
              X (I, J, K, 1) = XNew
            ENDIF
```

```
            IF (YNew.GT.C (Ptr + TF_UPPER_LIMIT_OUTPUT)) THEN
              Value = C (Ptr + TF_UPPER_LIMIT_OUTPUT)
            ELSEIF (YNew.LT.C (Ptr + TF_LOWER_LIMIT_OUTPUT)) THEN
              Value = C (Ptr + TF_LOWER_LIMIT_OUTPUT)
            ELSE
              Value = YNew
            ENDIF
   20     CONTINUE

          Out (I, J) = Value
   30 CONTINUE

      RETURN
      END
```

3.5. Thresholding

The thresholding algorithm is used to partition the image into points which are of interest and those that are not of interest. Pixels are zeroed if they are not of interest. A pixel is passed if the intensity is above a calculated lower threshold value and below a fixed upper threshold value. The lower threshold supports two of the modes which are in the Georgia Tech VLSI design. This includes a simple, fixed threshold and an adaptive threshold based on the average and first central absolute moment of the surrounding eight pixels.

3.6. Simple Thresholding

3.6.1. Data Fields

Action	Field Header	Field Data	Data Type
input	*Dimensions*	FPA dimension	Integer
input	*Simple Thresholding Limits*	Lower limit	Integer
		Upper limit	Integer
output	*Simple Thresholding Statistics*	Lower limit used	Integer
		Upper limit used	Integer
		Count of pixels exceeding limit	Integer
modify	*Pixel Data*	Pixel data array	Integer [1..Dimension] [1..Dimension]

3.6.2. Module Listing

```
      PROGRAM STHRESH
C
C     Simple Thresholding Test Module
C
C     conforms to the Ga. Tech Algorithm Evaluation Data Format
C
C     Fortran translation of Occam code
C     Steve Gieseking
C     Andrew Henshaw
C
C     Harness written by Andrew Henshaw     Jan 23, 1990
C     Using Microsoft Fortran
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
C
      CHARACTER*(*) Dim, Pixels, Limits
C
C     Valid Section Headers
C
      PARAMETER (Dim   ='Dimensions')
      PARAMETER (Pixels='Pixel Data')
      PARAMETER (Limits='Simple Thresholding Limits')
C
      PARAMETER (maxSize=128)     ! maximum FPA size

      INTEGER n, count, lower, upper
      INTEGER in(maxSize,maxSize), out(maxSize, maxSize)
      CHARACTER header*72
      LOGICAL runFlag
      DATA lower /0/              ! default values
      DATA upper /32767/

      WRITE (6,*) '% Processed by Simple Thresholding module.'
      runFlag = .TRUE.
      DO WHILE (runFlag)
        READ (5,1000) header
1000    FORMAT (A72)
        IF (header.EQ.Dim) THEN
          READ (5,*) n
          WRITE (6,*) Dim
          WRITE (6,*) n
        ELSE IF (header.EQ.Limits) THEN
          READ (5,*) lower, upper
```

```
          WRITE (6,*) Limits
          WRITE (6,*) lower, upper
        ELSE IF (header.EQ.Pixels) THEN
          READ (5,*) ((in(row,col),col=1,n),row=1,n)

          CALL SmpThrsh (n, lower, upper, count, in, out)

          WRITE (6,*) Pixels
          WRITE (6,*) ((out(row,col),col=1,n),row=1,n)
        ELSE IF (header.EQ.'End') THEN
          WRITE (6,*) 'End'
          runFlag = .FALSE.
        ELSE
          WRITE (6,*) header
        END IF
      END DO

      END

C***********************************************************************

      SUBROUTINE SmpThrsh (n, lower, upper, count, in, out)

      PARAMETER (maxSize=64)
      INTEGER n, lower, upper, count
      INTEGER in(maxSize, maxSize)
      INTEGER out(maxSize, maxSize)
      INTEGER row, col, pixel


      count = 0
      DO 30 row = 1, n
        DO 30 col = 1, n
          pixel = in(row,col)
          IF ((pixel.GE.lower).AND.(pixel.LE.upper)) THEN
            count = count + 1
            out(row,col) = pixel
          ELSE
            out(row,col) = 0
          END  IF
30    CONTINUE

C     Put Statistics onto data stream
      WRITE (6,*) 'Simple Thresholding Statistics'
```

```
      WRITE (6,*) lower, upper, count

      RETURN
      END
```

3.7. Adaptive Thresholding

3.7.1. Data Fields

Action	Field Header	Field Data	Data Type
input	*Dimensions*	FPA dimension	Integer
input	*Adaptive Thresholding Parameters*	Upper limit	Integer
		k1	Integer
		k2	Integer
		k3	Integer
		Scale	Integer
output	*Adaptive Thresholding Statistics*	Upper limit used	Integer
		Count of pixels exceeding limit	Integer
modify	*Pixel Data*	Pixel data array	Integer [1..Dimension] [1..Dimension]

3.7.2. Module Listing

```
      PROGRAM ADTHRESH
C
C     Adaptive Thresholding Test Module
C
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
C     400 Tenth St.  CRB 390
C     Atlanta, GA  30332-0540
C     Contact: Andrew Henshaw  (404)894-2521
C
C     conforms to the Ga. Tech Algorithm Evaluation Data Format
C
C     Fortran translation of Occam code
C     Steve Gieseking
C     Roy Melton
```

```
C     Harness written by A. M. Henshaw     Jan 23, 1990
C     Using Microsoft Fortran
C
      CHARACTER*(*) Dim, Pixels
      PARAMETER (Dim    ='Dimensions       ')
      PARAMETER (Pixels='Pixel Data       ')
      PARAMETER (Parms='Adaptive Thresholding Parameters')
      PARAMETER (maxSize=64)

      INTEGER n, count, k1, k2, k3, scale, sum, upper
      INTEGER in(maxSize,maxSize), out(maxSize, maxSize)
      CHARACTER header*72
      LOGICAL runFlag
      DATA k1 /1/
      DATA k2 /0/
      DATA k3 /0/
      DATA scale /8/
      DATA upper /32767/

      WRITE (6,*) '% Processed by Adaptive Thresholding module.'
      runFlag = .TRUE.
      DO WHILE (runFlag)
        READ (5,1000) header
1000    FORMAT (A72)
        IF (header.EQ.Dim) THEN
          READ (5,*) n
          WRITE (6,*) Dim
          WRITE (6,*) n
        ELSE IF (header.EQ.Parms) THEN
          READ (5,*) upper, k1, k2, k3, scale
          WRITE (6,*) Parms
          WRITE (6,*) upper, k1, k2, k3, scale
        ELSE IF (header.EQ.Pixels) THEN
          READ (5,*) ((in(row,col),col=1,n),row=1,n)

          CALL AdThrsh (n, upper, count, sum,
     +                 k1, k2, k3, scale, in, out)

          WRITE (6,*) Pixels
          WRITE (6,*) ((out(row,col),col=1,n),row=1,n)
        ELSE IF (header.EQ.'End') THEN
          WRITE (6,*) 'End'
          runFlag = .FALSE.
        ELSE
```

```
        WRITE (6,*) header
      END IF
    END DO

    END

C***********************************************************************

      SUBROUTINE AdThrsh (N, Upper, Count, Sum, K1, K2, K3, Scale,
     +                    In, Out)

      PARAMETER (maxSize=64)
      INTEGER N, Upper, Count, Sum, K1, K2, K3, Scale
      INTEGER In(maxSize, maxSize)
      INTEGER Out(maxSize, maxSize)
      INTEGER Average, I, J, K, L, Lower, Stat


      Count = 0
      Sum = 0

      DO 30 I = 1, N
        DO 30 J = 1, N
          IF (((I.EQ.1).OR.(I.EQ.N)).OR.((J.EQ.1).OR.(J.EQ.N))) THEN
            Out (I, J) = 0
          ELSE
            Average = In (I-1, J-1) + In (I-1, J) + In (I-1, J+1) +
     +                In (I, J-1)   + In (I, J+1) +
     +                In (I+1, J-1) + In (I+1, J) + In (I+1, J+1)
            Stat = 0
            DO 10 K = -1, 1
              DO 10 L = -1, 1
                IF ((K.NE.0).AND.(L.NE.0)) THEN
                  Stat = Stat + ABS ((In (I+K, J+L) * 8) - Average)
                ENDIF
   10       CONTINUE
            Lower = ((Average * K1) + (Stat * K2) + K3) / Scale
            Sum = Sum + Stat

            IF ((In (I, J).GE.Lower).AND.(In (I, J).LE.Upper)) THEN
              Out (I, J) = In (I, J)
              Count = Count + 1
            ELSE
              Out (I, J) = 0
```

```
          ENDIF
        ENDIF
   30 CONTINUE

C       Put Statistics onto data stream
        WRITE (6,*) 'Adaptive Thresholding Statistics'
        WRITE (6,*) Upper, Count

        RETURN
        END
```

3.8. Clustering & Centroiding

3.8.1. Description

The clustering algorithm forms connected sets of pixels based on the surrounding pixels. Two pixels are elements of the same cluster of pixels if they are one of the eight nearest neighbors of each other. The centroiding algorithm calculates the area centroid and the intensity weighted centroid of the clusters specified by the clustering algorithm.

3.8.2. Data Fields

Action	Field Header	Field Data	Data Type
input	*Dimensions*	FPA dimension	Integer
input	*Pixel Data*	Pixel data array	Integer [1..Dimension] [1..Dimension]
output	*Clusters*	Cluster count	Integer
output	*Centroids*	Vector of the following repeated Cluster count times	
		Area centroid (X)	Integer
		Area centroid (Y)	Integer
		Intensity centroid (X)	Integer
		Intensity centroid (Y)	Integer
		Area in pixels	Integer
		Total cluster intensity	Integer

3.8.3. Module Listing

```
PROGRAM CENTROID
```

```
C
C     Clustering and Centroiding Test Module
C
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
C     400 Tenth St.  CRB 390
C     Atlanta, GA  30332-0540
C     Contact: Andrew Henshaw  (404)894-2521
C
C     conforms to the Ga. Tech Algorithm Evaluation Data Format
C
C     Fortran translation of Occam code
C     Steve Gieseking
C     Roy W. Melton     Feb  12, 1990
C
C     Harness written by Andrew Henshaw     Jan 23, 1990
C     Using Microsoft Fortran
C
      CHARACTER*(*) Dim, Pixels
      PARAMETER (Dim   ='Dimensions      ')
      PARAMETER (Pixels='Pixel Data      ')
      PARAMETER (MAX_SIZE=64)
      PARAMETER (MAX_CLUSTERS=1024)

      INTEGER ClusterCount, N
      INTEGER Frame (MAX_SIZE, MAX_SIZE)
      INTEGER Clusters (MAX_CLUSTERS, 6)
      CHARACTER header*72
      LOGICAL runFlag

      WRITE (6,*) '% Processed by Centroid Image Module.'
      runFlag = .TRUE.
      DO WHILE (runFlag)
        READ (5,1000) header
1000    FORMAT (A72)
        IF (header.EQ.Dim) THEN
          READ (5,*) N
          WRITE (6,*) Dim
          WRITE (6,*) N
        ELSE IF (header.EQ.Pixels) THEN
          READ (5,*) ((Frame(row,col),col=1,N),row=1,N)

          CALL CentroidImage (Frame, Clusters, N, ClusterCount)
```

```
          WRITE (6,*) Pixels
          WRITE (6,*) ((Frame(row,col),col=1,N),row=1,N)
          WRITE (6,*) 'Clusters'
          WRITE (6,*) ClusterCount
          IF (ClusterCount.GT.0) THEN
            WRITE (6,*) 'Centroids'
            WRITE (6,*) ((Clusters (row, col), col=1,6),
     +                                        row=1,ClusterCount)
          ENDIF
        ELSE IF (header.EQ.'End') THEN
          WRITE (6,*) 'End'
          runFlag = .FALSE.
        ELSE
          WRITE (6,*) header
        END IF
      END DO

      END

C*********************************************************************

      SUBROUTINE CentroidImage (Frame, CData, N, ClusterCount)

      PARAMETER (MAX_SIZE=64)
      PARAMETER (MAX_CLUSTERS=1024)
      PARAMETER (CSum = 1)
      PARAMETER (CSumX = 2)
      PARAMETER (CSumY = 3)
      PARAMETER (ISum = 4)
      PARAMETER (ISumX = 5)
      PARAMETER (ISumY = 6)
      PARAMETER (ACoorX = 1)
      PARAMETER (ACoorY = 2)
      PARAMETER (ICoorX = 3)
      PARAMETER (ICoorY = 4)
      PARAMETER (Area = 5)
      PARAMETER (Intensity = 6)
      INTEGER Frame (MAX_SIZE, MAX_SIZE)
      INTEGER CData (MAX_CLUSTERS, 6)
      INTEGER N, ClusterCount
      INTEGER C0, C1, CNM1, CN, CNP1, FinalCluster, I, J
      INTEGER Cluster (MAX_SIZE + 1), Temp (6)
      INTEGER Reassign
      DIMENSION Reassign (0:MAX_CLUSTERS-1)
```

```
      DO 10 I=1,N+1
        Cluster (I) = 0
   10 CONTINUE
      FinalCluster = 0
      Reassign (0) = 0

      DO 50 I=1,N
C       /* Initialize Row */
        C1 = 0
        CNP1 = 0
        CN = Cluster (1)
   20   IF (CN.NE.Reassign (CN)) THEN
          CN = Reassign (CN)
          GOTO 20
        ENDIF
        DO 40 J = 1, N
          CNM1 = Cluster (J+1)
   30     IF (CNM1.NE.Reassign (CNM1)) THEN
            CNM1 = Reassign (CNM1)
            GOTO 30
          ENDIF
          IF (Frame (I, J).EQ.0) THEN
            C0 = 0
          ELSEIF (C1.NE.0) THEN
C           /* Add Pixel to Cluster */
            IF ((CNM1.NE.0).AND.(CNM1.NE.C1)) THEN
C             /* Merge C1, CNM1 */
              CData (C1, CSum) = CData (C1, CSum) +
     +                           CData (CNM1, CSum) + 1
              CData (C1, CSumX) = CData (C1, CSumX) +
     +                            CData (CNM1, CSumX) + J
              CData (C1, CSumY) = CData (C1, CSumY) +
     +                            CData (CNM1, CSumY) + I
              CData (C1, ISum) = CData (C1, ISum) +
     +                           CData (CNM1, ISum) + Frame (I, J)
              CData (C1, ISumX) = CData (C1, ISumX) +
     +                            CData (CNM1, ISumX) +
     +                            (Frame (I, J) * J)
              CData (C1, ISumY) = CData (C1, ISumY) +
     +                            CData (CNM1, ISumY) +
     +                            (Frame (I, J) * I)
              CData (CNM1, CSum) = 0
              Reassign (CNM1) = C1
```

```
                CNM1 = C1
                CN = Reassign (CN)
                C0 = C1

              ELSE
C                /* Add to C1 */
                CData (C1, CSum) = CData (C1, CSum) + 1
                CData (C1, CSumX) = CData (C1, CSumX) + J
                CData (C1, CSumY) = CData (C1, CSumY) + I
                CData (C1, ISum) = CData (C1, ISum) + Frame (I, J)
                CData (C1, ISumX) = CData (C1, ISumX) +
     +                                 (Frame (I, J) * J)
                CData (C1, ISumY) = CData (C1, ISumY) +
     +                                 (Frame (I, J) * I)
                C0 = C1
              ENDIF

            ELSEIF (CN.NE.0) THEN
C             /* Add to CN */
              CData (CN, CSum) = CData (CN, CSum) + 1
              CData (CN, CSumX) = CData (CN, CSumX) + J
              CData (CN, CSumY) = CData (CN, CSumY) + I
              CData (CN, ISum) = CData (CN, ISum) + Frame (I, J)
              CData (CN, ISumX) = CData (CN, ISumX) + (Frame (I, J) * J)
              CData (CN, ISumY) = CData (CN, ISumY) + (Frame (I, J) * I)
              C0 = CN

            ELSEIF (CNM1.NE.0) THEN
              IF ((CNP1.NE.0).AND.(CNP1.NE.CNM1)) THEN
C               /* Merge CNM1, CNP1 */
                CData (CNM1, CSum) = CData (CNM1, CSum) +
     +                               CData (CNP1, CSum) + 1
                CData (CNM1, CSumX) = CData (CNM1, CSumX) +
     +                                CData (CNP1, CSumX) + J
                CData (CNM1, CSumY) = CData (CNM1, CSumY) +
     +                                CData (CNP1, CSumY) + I
                CData (CNM1, ISum) = CData (CNM1, ISum) +
     +                               CData (CNP1, ISum) + Frame (I, J)
                CData (CNM1, ISumX) = CData (CNM1, ISumX) +
     +                                CData (CNP1, ISumX) +
     +                                (Frame (I, J) * J)
                CData (CNM1, ISumY) = CData (CNM1, ISumY) +
     +                                CData (CNP1, ISumY) +
     +                                (Frame (I, J) * I)
```

```
              CData (CNP1, CSum) = 0
              Reassign (CNP1) = CNM1
              C0 = CNM1

            ELSE
C             /* Add to CNM1 */
              CData (CNM1, CSum) = CData (CNM1, CSum) + 1
              CData (CNM1, CSumX) = CData (CNM1, CSumX) + J
              CData (CNM1, CSumY) = CData (CNM1, CSumY) + I
              CData (CNM1, ISum) = CData (CNM1, ISum) + Frame (I, J)
              CData (CNM1, ISumX) = CData (CNM1, ISumX) +
     +                              (Frame (I, J) * J)
              CData (CNM1, ISumY) = CData (CNM1, ISumY) +
     +                              (Frame (I, J) * I)
              C0 = CNM1
            ENDIF

          ELSEIF (CNP1.NE.0) THEN
C           /* Add to CNP1 */
            CData (CNP1, CSum) = CData (CNP1, CSum) + 1
            CData (CNP1, CSumX) = CData (CNP1, CSumX) + J
            CData (CNP1, CSumY) = CData (CNP1, CSumY) + I
            CData (CNP1, ISum) = CData (CNP1, ISum) + Frame (I, J)
            CData (CNP1, ISumX) = CData (CNP1, ISumX) +
     +                            (Frame (I, J) * J)
            CData (CNP1, ISumY) = CData (CNP1, ISumY) +
     +                            (Frame (I, J) * I)
            C0 = CNP1

          ELSE
C           /* New Cluster */
            FinalCluster = FinalCluster + 1
            C0 = FinalCluster
            CData (C0, CSum) = 1
            CData (C0, CSumX) = J
            CData (C0, CSumY) = I
            CData (C0, ISum) = Frame (I, J)
            CData (C0, ISumX) = Frame (I, J) * J
            CData (C0, ISumY) = Frame (I, J) * I
            Reassign (C0) = C0
          ENDIF

          Cluster (J) = C0
```

```
C         /* Update for next column */
          C1 = C0
          CNP1 = CN
          CN = CNM1

   40   CONTINUE
   50 CONTINUE

C     /* Output Centroids */
      ClusterCount = 0
      DO 70 I=1,FinalCluster
        IF (CData (I, CSum).NE.0) THEN
C         /* Valid Cluster */
          Temp (ACoorX) = (CData (I, CSumX) +
     +                ISHFT (CData (I, CSum), -1) ) / CData (I, CSum)
          Temp (ACoorY) = (CData (I, CSumY) +
     +                ISHFT (CData (I, CSum), -1) ) / CData (I, CSum)
          Temp (ICoorX) = (CData (I, ISumX) +
     +                ISHFT (CData (I, ISum), -1) ) / CData (I, ISum)
          Temp (ICoorY) = (CData (I, ISumY) +
     +                ISHFT (CData (I, ISum), -1) ) / CData (I, ISum)
          Temp (Area) = CData (I, CSum)
          Temp (Intensity) = CData (I, ISum)
          ClusterCount = ClusterCount + 1
          DO 60 J=1,6
            CData (ClusterCount, J) = Temp (J)
   60     CONTINUE
        ENDIF

   70 CONTINUE

      RETURN
      END
```

3.9. Object Processing

The Object Processing module uses a Kalman Filter to track centroids as they appear in the data stream. The output of the program is a list of objects that have been tracked for some number of time periods and their positions.

3.9.1. Description

3.9.2. Listing

```
      PROGRAM VER1
```

```
C
C     Tracking Test Module
C
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
C     400 Tenth St.  CRB 390
C     Atlanta, GA 30332-0540
C     Contact:
C
C     Conforms to the Ga. Tech Algorithm Evaluation Data Format
C
C     FORTRAN translation of Occam code
C     Steven R. Gieseking
C     Roy W. Melton     Feb 14, 1990
C
C     Harness written by Andrew M. Henshaw
C     Using Microsoft FORTRAN
C
      PARAMETER (ARRAY_SIZE = 64)
      PARAMETER (RADIANS_PER_PIXEL = 150.0E-6)
      PARAMETER (MAX_TRACKS = 32)
      PARAMETER (MAX_TRACKSP1 = 33)
      PARAMETER (MAX_TRACKSP2 = 34)
      PARAMETER (MAX_DIST = 6.0)
      PARAMETER (MAX_MISS = 2)

      CHARACTER*(*) DelT, Centroid, Cluster
      PARAMETER (DelT     = 'Delta T         ')
      PARAMETER (Centroid = 'Centroids       ')
      PARAMETER (Cluster  = 'Clusters        ')
      CHARACTER Header*72

C     /* TC Record from Pascal Version */
      INTEGER TCTrackNum     (MAX_TRACKSP2)
      INTEGER TCPriority     (MAX_TRACKSP2)
      REAL    TCRPriority    (MAX_TRACKSP2)
      REAL    TCPositionX    (MAX_TRACKSP2)
      REAL    TCPositionY    (MAX_TRACKSP2)
      REAL    TCWeightX      (MAX_TRACKSP2)
      REAL    TCWeightY      (MAX_TRACKSP2)
      REAL    TCMinDistance  (MAX_TRACKSP2)
      INTEGER TCNumCor       (MAX_TRACKSP2)
      INTEGER TCCenID        (MAX_TRACKSP2)
      REAL    TCCenAcoorX    (MAX_TRACKSP2)
```

```
      REAL     TCCenAcoorY      (MAX_TRACKSP2)
      REAL     TCCenIcoorX      (MAX_TRACKSP2)
      REAL     TCCenIcoorY      (MAX_TRACKSP2)
      INTEGER TCCenArea        (MAX_TRACKSP2)
      INTEGER TCCenIntensity (MAX_TRACKSP2)

C     /* TF Record from Pascal Version */
      INTEGER TFTrackID     (MAX_TRACKSP1)
      INTEGER TFNumCor      (MAX_TRACKSP1)
      INTEGER TFNumMiss     (MAX_TRACKSP1)
      REAL     TFEstimateX  (MAX_TRACKSP1)
      REAL     TFEstimateVX (MAX_TRACKSP1)
      REAL     TFXP11       (MAX_TRACKSP1)
      REAL     TFXP12       (MAX_TRACKSP1)
      REAL     TFXP22       (MAX_TRACKSP1)
      REAL     TFEstimateY  (MAX_TRACKSP1)
      REAL     TFEstimateVY (MAX_TRACKSP1)
      REAL     TFYP11       (MAX_TRACKSP1)
      REAL     TFYP12       (MAX_TRACKSP1)
      REAL     TFYP22       (MAX_TRACKSP1)
      INTEGER TFIntensity  (MAX_TRACKSP1)
      INTEGER TFCSO        (MAX_TRACKSP1)

      LOGICAL RunFlag
      INTEGER Clusters
      INTEGER I
      INTEGER LastTrack, LastNew
      INTEGER TrackID

      REAL DT, DT2, DTSqr
      REAL RadPix2, PixelsPerRadian, ProcessNoise, MeasurementNoise
      REAL InitialP11, InitialP12, InitialP22
      REAL PixelOffset


      COMMON /Centroid/ CID,CAcoorX, CAcoorY, CIcoorX, CIcoorY,
     +                  CArea, CIntensity,
     +                  CentroidID, PixelOffset

      COMMON /DeltaT/ DT, DT2, DTSqr
      COMMON /Initial/ InitialP11, InitialP12, InitialP22
      COMMON /Last/ LastNew, LastTrack
      COMMON /Measure/ MeasurementNoise
      COMMON /Pixel/ PixelsPerRadian, RadPix2, TrackID
```

```
      COMMON /Process/ ProcessNoise
      COMMON /TC/ TCTrackNum, TCPriority,
     +            TCRPriority,
     +            TCPositionX,
     +            TCPositionY,
     +            TCWeightX, TCWeightY,
     +            TCMinDistance,
     +            TCNumCor, TCCenID,
     +            TCCenAcoorX,
     +            TCCenAcoorY,
     +            TCCenIcoorX,
     +            TCCenIcoorY,
     +            TCCenArea,
     +            TCCenIntensity

      COMMON /TF/ TFTrackID, TFNumCor,
     +            TFNumMiss,
     +            TFEstimateX,
     +            TFEstimateVX,
     +            TFXP11, TFXP12,
     +            TFXP22,
     +            TFEstimateY,
     +            TFEstimateVY,
     +            TFYP11, TFYP12,
     +            TFYP22,
     +            TFIntensity, TFCSO

      RadPix2 = RADIANS_PER_PIXEL * RADIANS_PER_PIXEL
      PixelsPerRadian = 1.0 / RADIANS_PER_PIXEL
      ProcessNoise = 2.5 * RadPix2
      MeasurementNoise = 1.0 * RadPix2

      InitialP11 =   2.0 * RadPix2
      InitialP12 =   0.0 * RadPix2
      InitialP22 = 100.0 * RadPix2

      PixelOffset = (ARRAY_SIZE / 2.0) - 0.5

C     /* Initialize */
      LastTrack = 1
      LastNew = 1
      DO 10 I = 1, MAX_TRACKSP2
        TCTrackNum (I) = I
        TCPriority (I) = 0
```

```
   10 CONTINUE
      TrackID = 1

      WRITE (6, *) '% Processed by Tracking Module'
      RunFlag = .TRUE.
      DO WHILE (RunFlag)
        READ (5, 1007) Header
        IF (Header.EQ.DelT) THEN
          CALL GetDeltaT (DT, DT2, DTSqr)
          WRITE (6, *) DelT
          WRITE (6, *) DT
        ELSEIF (Header.EQ.Cluster) THEN
          READ (5, *) Clusters
          WRITE (6, *) Cluster
          WRITE (6, *) Clusters
        ELSEIF (Header.EQ.Centroid) THEN
          WRITE (6, *) Centroid
          CALL TrackFrame (Clusters)
        ELSEIF (Header.EQ.'End') THEN
          WRITE (6, *) 'End'
          RunFlag = .FALSE.
        ELSE
          WRITE (6, *) Header
        ENDIF
      ENDDO

 1000 FORMAT (A10)
 1001 FORMAT (A)
 1002 FORMAT (1X, A21, 3I4, 4F10.10)
 1003 FORMAT (1X, '&&& lastnew,lasttrack : ', I4, ' ', I4)
 1004 FORMAT (1X, '&&& lasttrack: ', I4)
 1005 FORMAT (1X, 'Track=', I4, ' ID=', I4, ' Pos=', 1P, E10.3,
     +        ' Vel=', E10.3, 0P)
 1006 FORMAT (1X, 'Frame ', I4, ' OK')
 1007 FORMAT (A72)
 1008 FORMAT (1X, 3I4, 1P, 4E13.3, 0P)
      END

C*************************************************************************
      SUBROUTINE GetDeltaT (DT, DT2, DTSqr)

      REAL DT, DT2, DTSqr

      READ (5, *) DT
```

```
      DT2 = 2.0 * DT
      DTSqr = DT * DT

      RETURN
      END

C**********************************************************************
      SUBROUTINE InitializeNewTracks (TCCenAcoorX, TCCenAcoorY,
     +                          TCPriority, TCTrackNum, TrackID)

      PARAMETER (MAX_TRACKSP1 = 33)
      PARAMETER (MAX_TRACKSP2 = 34)
      REAL    TCCenAcoorX     (MAX_TRACKSP2)
      REAL    TCCenAcoorY     (MAX_TRACKSP2)
      INTEGER TCPriority      (MAX_TRACKSP2)
      INTEGER TCTrackNum      (MAX_TRACKSP2)
      INTEGER TrackID
      REAL InitialP11, InitialP12, InitialP22
      INTEGER LastNew, LastTrack

      INTEGER TFTrackID     (MAX_TRACKSP1)
      INTEGER TFNumCor      (MAX_TRACKSP1)
      INTEGER TFNumMiss     (MAX_TRACKSP1)
      REAL    TFEstimateX   (MAX_TRACKSP1)
      REAL    TFEstimateVX  (MAX_TRACKSP1)
      REAL    TFXP11        (MAX_TRACKSP1)
      REAL    TFXP12        (MAX_TRACKSP1)
      REAL    TFXP22        (MAX_TRACKSP1)
      REAL    TFEstimateY   (MAX_TRACKSP1)
      REAL    TFEstimateVY  (MAX_TRACKSP1)
      REAL    TFYP11        (MAX_TRACKSP1)
      REAL    TFYP12        (MAX_TRACKSP1)
      REAL    TFYP22        (MAX_TRACKSP1)
      INTEGER TFIntensity   (MAX_TRACKSP1)
      INTEGER TFCSO         (MAX_TRACKSP1)

      COMMON /Initial/ InitialP11, InitialP12, InitialP22
      COMMON /Last/ LastNew, LastTrack
      COMMON /TF/ TFTrackID, TFNumCor,
     +            TFNumMiss,
     +            TFEstimateX,
     +            TFEstimateVX,
     +            TFXP11, TFXP12,
     +            TFXP22,
```

```
     +            TFEstimateY,
     +            TFEstimateVY,
     +            TFYP11, TFYP12,
     +            TFYP22,
     +            TFIntensity, TFCSO

      INTEGER I, Ptr

      DO 310 I = LastTrack, (LastNew - 1)
        Ptr = TCTrackNum (I)
        TFTrackID (Ptr) = TrackID
        TrackID = TrackID + 1
        TFNumCor (Ptr) = 1
        TFNumMiss (Ptr) = 0
        TCPriority (I) = 1
        TFEstimateX (Ptr) = TCCenAcoorX (I)
        TFEstimateVX (Ptr) = 0.0
        TFXP11 (Ptr) = InitialP11
        TFXP12 (Ptr) = InitialP12
        TFXP22 (Ptr) = InitialP22
        TFEstimateY (Ptr) = TCCenAcoorY (I)
        TFEstimateVY (Ptr) = 0.0
        TFYP11 (Ptr) = InitialP11
        TFYP12 (Ptr) = InitialP12
        TFYP22 (Ptr) = InitialP22
  310 CONTINUE
      RETURN
      END

C***********************************************************************
      SUBROUTINE InputCentroid

      PARAMETER (RADIANS_PER_PIXEL = 150.0E-6)
      INTEGER CID
      REAL    CAcoorX
      REAL    CAcoorY
      REAL    CIcoorX
      REAL    CIcoorY
      INTEGER CArea
      INTEGER CIntensity
      INTEGER CentroidID
      REAL    PixelOffset
      COMMON  /Centroid/ CID,CAcoorX, CAcoorY, CIcoorX, CIcoorY,
     +                   CArea, CIntensity,
```

```
     +                   CentroidID, PixelOffset

      INTEGER IAcoorX, IAcoorY, IICoorX, IICoorY

      READ (5, *) IAcoorX, IAcoorY, IIcoorX, IIcoorY,
     +            CArea, CIntensity
      WRITE (6, *) IAcoorX, IAcoorY, IIcoorX, IIcoorY,
     +             CArea, CIntensity
      CID = CentroidID
      CentroidID = CentroidID + 1

      CAcoorX = (IAcoorX - PixelOffset) * RADIANS_PER_PIXEL
      CAcoorY = (PixelOffset - IAcoorY) * RADIANS_PER_PIXEL
      CIcoorX = (IIcoorX - PixelOffset) * RADIANS_PER_PIXEL
      CIcoorY = (PixelOffset - IIcoorY) * RADIANS_PER_PIXEL

      RETURN
      END

C**********************************************************************
      SUBROUTINE KalmanMeasurementUpdate (MeasuredPosition,
     +                                    Position, Velocity,
     +                                    P11, P12, P22      )

      REAL MeasuredPosition, Velocity, P11, P12, P22
      REAL MeasurementNoise
      COMMON /Measure/ MeasurementNoise

      REAL K1, K2, StateError, Temp

      StateError = MeasuredPosition - Position
      Temp = 1.0 / (P11 + MeasurementNoise)
      K1 = Temp * P11
      K2 = Temp * P12
      Temp = P12
      P11 = (1.0 - K1) * P11
      P12 = (1.0 - K1) * P12
      P22 = P22 - (K2 * Temp)
      Position = Position + (K1 * StateError)
      Velocity = Velocity + (K2 * StateError)

      RETURN
      END
```

```
C***********************************************************************
      SUBROUTINE KalmanTimeUpdate (Position, Velocity, P11, P12, P22)

      REAL Position, Velocity, P11, P12, P22

      REAL DT, DT2, DTSqr, ProcessNoise
      COMMON /DeltaT/ DT, DT2, DTSqr
      COMMON /Process/ ProcessNoise

      Position = Position + (DT * Velocity)
      P11 = P11 + (DT2 * P12) + (DTSqr * P22)
      P12 = P12 + (DT * P22)
      P22 = P22 + ProcessNoise
      RETURN
      END

C***********************************************************************
      SUBROUTINE RankTracks

      PARAMETER (MAX_TRACKSP1 = 33)
      PARAMETER (MAX_TRACKSP2 = 34)
      REAL    DT, DT2, DTSqr, ProcessNoise
      INTEGER LastNew, LastTrack
      INTEGER TCTrackNum     (MAX_TRACKSP2)
      INTEGER TCPriority     (MAX_TRACKSP2)
      REAL    TCRPriority    (MAX_TRACKSP2)
      REAL    TCPositionX    (MAX_TRACKSP2)
      REAL    TCPositionY    (MAX_TRACKSP2)
      REAL    TCWeightX      (MAX_TRACKSP2)
      REAL    TCWeightY      (MAX_TRACKSP2)
      REAL    TCMinDistance  (MAX_TRACKSP2)
      INTEGER TCNumCor       (MAX_TRACKSP2)
      INTEGER TCCenID        (MAX_TRACKSP2)
      REAL    TCCenAcoorX    (MAX_TRACKSP2)
      REAL    TCCenAcoorY    (MAX_TRACKSP2)
      REAL    TCCenIcoorX    (MAX_TRACKSP2)
      REAL    TCCenIcoorY    (MAX_TRACKSP2)
      INTEGER TCCenArea      (MAX_TRACKSP2)
      INTEGER TCCenIntensity (MAX_TRACKSP2)
      INTEGER TFTrackID    (MAX_TRACKSP1)
      INTEGER TFNumCor     (MAX_TRACKSP1)
      INTEGER TFNumMiss    (MAX_TRACKSP1)
      REAL    TFEstimateX  (MAX_TRACKSP1)
      REAL    TFEstimateVX (MAX_TRACKSP1)
```

```
      REAL    TFXP11         (MAX_TRACKSP1)
      REAL    TFXP12         (MAX_TRACKSP1)
      REAL    TFXP22         (MAX_TRACKSP1)
      REAL    TFEstimateY    (MAX_TRACKSP1)
      REAL    TFEstimateVY   (MAX_TRACKSP1)
      REAL    TFYP11         (MAX_TRACKSP1)
      REAL    TFYP12         (MAX_TRACKSP1)
      REAL    TFYP22         (MAX_TRACKSP1)
      INTEGER TFIntensity    (MAX_TRACKSP1)
      INTEGER TFCSO          (MAX_TRACKSP1)

      COMMON /DeltaT/ DT, DT2, DTSqr
      COMMON /Last/ LastNew, LastTrack
      COMMON /Process/ ProcessNoise
      COMMON /TC/ TCTrackNum, TCPriority,
     +          TCRPriority,
     +          TCPositionX,
     +          TCPositionY,
     +          TCWeightX, TCWeightY,
     +          TCMinDistance,
     +          TCNumCor, TCCenID,
     +          TCCenAcoorX,
     +          TCCenAcoorY,
     +          TCCenIcoorX,
     +          TCCenIcoorY,
     +          TCCenArea,
     +          TCCenIntensity
      COMMON /TF/ TFTrackID, TFNumCor,
     +          TFNumMiss,
     +          TFEstimateX,
     +          TFEstimateVX,
     +          TFXP11, TFXP12,
     +          TFXP22,
     +          TFEstimateY,
     +          TFEstimateVY,
     +          TFYP11, TFYP12,
     +          TFYP22,
     +          TFIntensity, TFCSO

      INTEGER I, J, Temp1, Temp2, Ptr

C     WRITE (6, 1003) LastNew, LastTrack
      IF (LastNew.GT.2) THEN
C       /* Sort on Priority */
```

```
      DO 520 I = 1, (LastNew - 2)
        Ptr = I
        DO 510 J = (I + 1), (LastNew - 1)
          IF (TCPriority (Ptr).LT.TCPriority (J)) THEN
            Ptr = J
          ENDIF
 510    CONTINUE
        IF (Ptr.NE.I) THEN
          Temp1 = TCTrackNum (Ptr)
          Temp2 = TCPriority (Ptr)
          TCTrackNum (Ptr) = TCTrackNum (I)
          TCPriority (Ptr) = TCPriority (I)
          TCTrackNum (I) = Temp1
          TCPriority (I) = Temp2
        ENDIF
 520  CONTINUE
    ENDIF

    LastTrack = 1
 530 IF (TCPriority (LastTrack).NE.0) THEN
      LastTrack = LastTrack + 1
      GOTO 530
    ENDIF
    LastNew = LastTrack
C   WRITE (6, 1004) LastTrack
    CALL UpdateTime (TCTrackNum)

1003 FORMAT (1X, '&&& lastnew,lasttrack : ', I4, ' ', I4)
1004 FORMAT (1X, '&&& lasttrack: ', I4)

    RETURN
    END

C***********************************************************************
    SUBROUTINE SetUpCorrelation

    PARAMETER (MAX_TRACKSP1 = 33)
    PARAMETER (MAX_TRACKSP2 = 34)
    PARAMETER (MAX_DIST = 6.0)
    INTEGER LastTrack, LastNew
    INTEGER TCTrackNum     (MAX_TRACKSP2)
    INTEGER TCPriority     (MAX_TRACKSP2)
    REAL    TCRPriority    (MAX_TRACKSP2)
    REAL    TCPositionX    (MAX_TRACKSP2)
```

```
      REAL    TCPositionY     (MAX_TRACKSP2)
      REAL    TCWeightX       (MAX_TRACKSP2)
      REAL    TCWeightY       (MAX_TRACKSP2)
      REAL    TCMinDistance   (MAX_TRACKSP2)
      INTEGER TCNumCor        (MAX_TRACKSP2)
      INTEGER TCCenID         (MAX_TRACKSP2)
      REAL    TCCenAcoorX     (MAX_TRACKSP2)
      REAL    TCCenAcoorY     (MAX_TRACKSP2)
      REAL    TCCenIcoorX     (MAX_TRACKSP2)
      REAL    TCCenIcoorY     (MAX_TRACKSP2)
      INTEGER TCCenArea       (MAX_TRACKSP2)
      INTEGER TCCenIntensity  (MAX_TRACKSP2)
      INTEGER TFTrackID     (MAX_TRACKSP1)
      INTEGER TFNumCor      (MAX_TRACKSP1)
      INTEGER TFNumMiss     (MAX_TRACKSP1)
      REAL    TFEstimateX   (MAX_TRACKSP1)
      REAL    TFEstimateVX  (MAX_TRACKSP1)
      REAL    TFXP11        (MAX_TRACKSP1)
      REAL    TFXP12        (MAX_TRACKSP1)
      REAL    TFXP22        (MAX_TRACKSP1)
      REAL    TFEstimateY   (MAX_TRACKSP1)
      REAL    TFEstimateVY  (MAX_TRACKSP1)
      REAL    TFYP11        (MAX_TRACKSP1)
      REAL    TFYP12        (MAX_TRACKSP1)
      REAL    TFYP22        (MAX_TRACKSP1)
      INTEGER TFIntensity   (MAX_TRACKSP1)
      INTEGER TFCSO         (MAX_TRACKSP1)

      COMMON /Last/ LastNew, LastTrack
      COMMON /TC/ TCTrackNum, TCPriority,
     +            TCRPriority,
     +            TCPositionX,
     +            TCPositionY,
     +            TCWeightX, TCWeightY,
     +            TCMinDistance,
     +            TCNumCor, TCCenID,
     +            TCCenAcoorX,
     +            TCCenAcoorY,
     +            TCCenIcoorX,
     +            TCCenIcoorY,
     +            TCCenArea,
     +            TCCenIntensity
      COMMON /TF/ TFTrackID, TFNumCor,
     +            TFNumMiss,
```

```
     +              TFEstimateX,
     +              TFEstimateVX,
     +              TFXP11, TFXP12,
     +              TFXP22,
     +              TFEstimateY,
     +              TFEstimateVY,
     +              TFYP11, TFYP12,
     +              TFYP22,
     +              TFIntensity, TFCSO

      INTEGER I, Ptr

      DO 610 I = 1, (LastTrack -1)
        Ptr = TCTrackNum (I)
        TCPositionX (I) = TFEstimateX (Ptr)
        TCPositionY (I) = TFEstimateY (Ptr)
        TCWeightX (I) = 1.0 / TFXP11 (Ptr)
        TCWeightY (I) = 1.0 / TFYP11 (Ptr)
        TCMinDistance (I) = MAX_DIST
        TCNumCor (I) = 0
  610 CONTINUE

      RETURN
      END

C**********************************************************************
      SUBROUTINE TrackFrame (Clusters)
      PARAMETER (ARRAY_SIZE = 64)
      PARAMETER (RADIANS_PER_PIXEL = 150.0E-6)
      PARAMETER (MAX_TRACKS = 32)
      PARAMETER (MAX_TRACKSP1 = 33)
      PARAMETER (MAX_TRACKSP2 = 34)
      PARAMETER (MAX_DIST = 6.0)
      PARAMETER (MAX_MISS = 2)
      INTEGER Clusters

C     /* TC Record from Pascal Version */
      INTEGER TCTrackNum      (MAX_TRACKSP2)
      INTEGER TCPriority      (MAX_TRACKSP2)
      REAL    TCRPriority     (MAX_TRACKSP2)
      REAL    TCPositionX     (MAX_TRACKSP2)
      REAL    TCPositionY     (MAX_TRACKSP2)
      REAL    TCWeightX       (MAX_TRACKSP2)
      REAL    TCWeightY       (MAX_TRACKSP2)
```

```
      REAL    TCMinDistance  (MAX_TRACKSP2)
      INTEGER TCNumCor       (MAX_TRACKSP2)
      INTEGER TCCenID        (MAX_TRACKSP2)
      REAL    TCCenAcoorX    (MAX_TRACKSP2)
      REAL    TCCenAcoorY    (MAX_TRACKSP2)
      REAL    TCCenIcoorX    (MAX_TRACKSP2)
      REAL    TCCenIcoorY    (MAX_TRACKSP2)
      INTEGER TCCenArea      (MAX_TRACKSP2)
      INTEGER TCCenIntensity (MAX_TRACKSP2)

C     /* TF Record from Pascal Version */
      INTEGER TFTrackID    (MAX_TRACKSP1)
      INTEGER TFNumCor     (MAX_TRACKSP1)
      INTEGER TFNumMiss    (MAX_TRACKSP1)
      REAL    TFEstimateX  (MAX_TRACKSP1)
      REAL    TFEstimateVX (MAX_TRACKSP1)
      REAL    TFXP11       (MAX_TRACKSP1)
      REAL    TFXP12       (MAX_TRACKSP1)
      REAL    TFXP22       (MAX_TRACKSP1)
      REAL    TFEstimateY  (MAX_TRACKSP1)
      REAL    TFEstimateVY (MAX_TRACKSP1)
      REAL    TFYP11       (MAX_TRACKSP1)
      REAL    TFYP12       (MAX_TRACKSP1)
      REAL    TFYP22       (MAX_TRACKSP1)
      INTEGER TFIntensity  (MAX_TRACKSP1)
      INTEGER TFCSO        (MAX_TRACKSP1)

C     /* Centroid Record from Pascal Version */
      INTEGER CID
      REAL    CAcoorX
      REAL    CAcoorY
      REAL    CIcoorX
      REAL    CIcoorY
      INTEGER CArea
      INTEGER CIntensity

      INTEGER LastTrack, LastNew
      INTEGER TrackID, CentroidID
      INTEGER Ptr, LowPtr
      REAL    DX, DY, Dist2, Priority, LowPriority
      LOGICAL NotMatchCentroid
      INTEGER LastCentroid

      REAL    DT, DT2, DTSqr
```

```
      REAL RadPix2, PixelsPerRadian, ProcessNoise, MeasurementNoise
      REAL InitialP11, InitialP12, InitialP22
      REAL PixelOffset

      INTEGER I
C     LOGICAL Error

      COMMON /Centroid/ CID,CAcoorX, CAcoorY, CIcoorX, CIcoorY,
     +                  CArea, CIntensity,
     +                  CentroidID, PixelOffset

      COMMON /DeltaT/ DT, DT2, DTSqr
      COMMON /Initial/ InitialP11, InitialP12, InitialP22
      COMMON /Last/ LastNew, LastTrack
      COMMON /Measure/ MeasurementNoise
      COMMON /Pixel/ PixelsPerRadian, RadPix2, TrackID
      COMMON /Process/ ProcessNoise
      COMMON /TC/ TCTrackNum, TCPriority,
     +            TCRPriority,
     +            TCPositionX,
     +            TCPositionY,
     +            TCWeightX, TCWeightY,
     +            TCMinDistance,
     +            TCNumCor, TCCenID,
     +            TCCenAcoorX,
     +            TCCenAcoorY,
     +            TCCenIcoorX,
     +            TCCenIcoorY,
     +            TCCenArea,
     +            TCCenIntensity

      COMMON /TF/ TFTrackID, TFNumCor,
     +            TFNumMiss,
     +            TFEstimateX,
     +            TFEstimateVX,
     +            TFXP11, TFXP12,
     +            TFXP22,
     +            TFEstimateY,
     +            TFEstimateVY,
     +            TFYP11, TFYP12,
     +            TFYP22,
     +            TFIntensity, TFCSO

C     Error = .FALSE.
```

```
C     WRITE (6, *) 'Start Tracking'

C     /* Perform Tracking */
      LastCentroid = 1
      CentroidId = 1

      DO 140 I = 1, Clusters
        CALL InputCentroid
C       /* Compare against tracks */
        NotMatchCentroid = .TRUE.
        Ptr = 1
  120   IF (Ptr.LT.LastTrack) THEN
          DX = CAcoorX - TCPositionX (Ptr)
          DY = CAcoorY - TCPositionY (Ptr)
          Dist2 = (TCWeightX (Ptr) * DX * DX) +
     +            (TCWeightY (Ptr) * DY * DY)
          IF (Dist2.LT.MAX_DIST) THEN
C           /* Correlated with this track file */
            TCNumCor (Ptr) = TCNumCor (Ptr) + 1
            NotMatchCentroid = .FALSE.
            IF (Dist2.LT.TCMinDistance (Ptr)) THEN
              TCMinDistance (Ptr) = Dist2
              TCCenID (Ptr) = CID
              TCCenAcoorX (Ptr) = CAcoorX
              TCCenAcoorY (Ptr) = CAcoorY
              TCCenIcoorX (Ptr) = CIcoorX
              TCCenIcoorY (Ptr) = CIcoorY
              TCCenArea (Ptr) = CArea
              TCCenIntensity (Ptr) = CIntensity
            ENDIF
          ENDIF
          Ptr = Ptr + 1
          GOTO 120
        ENDIF

C       /* Compare against uncorrelated centroids */
        IF (NotMatchCentroid.AND.(Ptr.LT.MAX_TRACKSP1)) THEN
C         /* Check to start new track */
          Priority = (CAcoorX * CAcoorX) + (CAcoorY * CAcoorY)
          IF (LastNew.LT.MAX_TRACKSP1) THEN
C           /* Room to start new track */
            TCRPriority (LastNew) = Priority
            TCCenID (LastNew) = CID
```

```
            TCCenAcoorX (LastNew) = CAcoorX
            TCCenAcoorY (LastNew) = CAcoorY
            TCCenIcoorX (LastNew) = CIcoorX
            TCCenIcoorY (LastNew) = CIcoorY
            TCCenArea (LastNew) = CArea
            TCCenIntensity (LastNew) = CIntensity
            LastNew = LastNew + 1
          ELSE
C           /* Check other new track for priority */
            LowPriority = -1.0
            LowPtr = LastNew
  130       IF (Ptr.LT.MAX_TRACKSP1) THEN
              IF (LowPriority.LT.TCPriority (Ptr)) THEN
                LowPtr = Ptr
                LowPriority = TCPriority (Ptr)
              ENDIF
              Ptr = Ptr + 1
              GOTO 130
            ENDIF
            IF (Priority.LT.LowPriority) THEN
              TCRPriority (LowPtr) = Priority
              TCCenID (LowPtr) = CID
              TCCenAcoorX (LowPtr) = CAcoorX
              TCCenAcoorY (LowPtr) = CAcoorY
              TCCenIcoorX (LowPtr) = CIcoorX
              TCCenIcoorY (LowPtr) = CIcoorY
              TCCenArea (LowPtr) = CArea
              TCCenIntensity (LowPtr) = CIntensity
            ENDIF
          ENDIF
        ENDIF
  140 CONTINUE

      CALL UpdateTracks (LastTrack)
      CALL InitializeNewTracks (TCCenAcoorX, TCCenAcoorY,
     +                   TCPriority, TCTrackNum, TrackID)
      CALL RankTracks
      CALL SetUpCorrelation
C     /* End PerformTracking */

C     CALL CheckTracks (TCTrackNum, Error)
C     WRITE (6, *) 'Ending Tracking'
      WRITE (6, *) 'Track Count'
      WRITE (6, *) (LastTrack - 1)
```

```
      WRITE (6, *) 'Tracks'
      DO 150 I = 1, (LastTrack - 1)
        Ptr = TCTrackNum (I)
        WRITE (6, 1008) TFTrackID (Ptr), TFNumCor (Ptr),
     +        TFNumMiss (Ptr), TFEstimateX (Ptr), TFEstimateVX (Ptr),
     +        TFEstimateY (Ptr), TFEstimateVY (Ptr)
  150 CONTINUE

 1008 FORMAT (1X, 3I4, 1P, 4E13.3, 0P)
      END

C************************************************************************
      SUBROUTINE UpdateTime (TCTrackNum)

      PARAMETER (MAX_TRACKSP1 = 33)
      PARAMETER (MAX_TRACKSP2 = 34)
      INTEGER TCTrackNum (MAX_TRACKSP2)

      REAL DT, DT2, DTSqr, ProcessNoise
      INTEGER LastNew, LastTrack
      INTEGER TFTrackID    (MAX_TRACKSP1)
      INTEGER TFNumCor     (MAX_TRACKSP1)
      INTEGER TFNumMiss    (MAX_TRACKSP1)
      REAL    TFEstimateX  (MAX_TRACKSP1)
      REAL    TFEstimateVX (MAX_TRACKSP1)
      REAL    TFXP11       (MAX_TRACKSP1)
      REAL    TFXP12       (MAX_TRACKSP1)
      REAL    TFXP22       (MAX_TRACKSP1)
      REAL    TFEstimateY  (MAX_TRACKSP1)
      REAL    TFEstimateVY (MAX_TRACKSP1)
      REAL    TFYP11       (MAX_TRACKSP1)
      REAL    TFYP12       (MAX_TRACKSP1)
      REAL    TFYP22       (MAX_TRACKSP1)
      INTEGER TFIntensity  (MAX_TRACKSP1)
      INTEGER TFCSO        (MAX_TRACKSP1)

      COMMON /DeltaT/ DT, DT2, DTSqr
      COMMON /Last/ LastNew, LastTrack
      COMMON /Process/ ProcessNoise
      COMMON /TF/ TFTrackID, TFNumCor,
     +            TFNumMiss,
     +            TFEstimateX,
     +            TFEstimateVX,
     +            TFXP11, TFXP12,
```

```
     +             TFXP22,
     +             TFEstimateY,
     +             TFEstimateVY,
     +             TFYP11, TFYP12,
     +             TFYP22,
     +             TFIntensity, TFCSO

      INTEGER I, Ptr

      DO 710 I = 1, (LastTrack - 1)
        Ptr = TCTrackNum (I)
C       WRITE (6, 1005) TCTrackNum (I), TFTrackID (Ptr),
C    +                  TFEstimateX (Ptr), TFEstimateVX (Ptr)
        CALL KalmanTimeUpdate (TFEstimateX (Ptr), TFEstimateVX (Ptr),
     +                         TFXP11 (Ptr), TFXP12 (Ptr), TFXP22 (Ptr))
        CALL KalmanTimeUpdate (TFEstimateY (Ptr), TFEstimateVY (Ptr),
     +                         TFYP11 (Ptr), TFYP12 (Ptr), TFYP22 (Ptr))
C       WRITE (6, 1005) TCTrackNum (I), TFTrackID (Ptr),
C    +                  TFEstimateX (Ptr), TFEstimateVX (Ptr)
  710 CONTINUE

 1005 FORMAT (1X, 'Track=', I4, ' ID=', I4, ' Pos=', 1P, E10.3,
     +        ' Vel=', E10.3, 0P)

      RETURN
      END

C***********************************************************************
      SUBROUTINE UpdateTracks (LastTrack)

      PARAMETER (MAX_TRACKSP1 = 33)
      PARAMETER (MAX_TRACKSP2 = 34)
      PARAMETER (MAX_MISS = 2)
      INTEGER LastTrack
      REAL MeasurementNoise

      INTEGER TCTrackNum     (MAX_TRACKSP2)
      INTEGER TCPriority     (MAX_TRACKSP2)
      REAL    TCRPriority    (MAX_TRACKSP2)
      REAL    TCPositionX    (MAX_TRACKSP2)
      REAL    TCPositionY    (MAX_TRACKSP2)
      REAL    TCWeightX      (MAX_TRACKSP2)
      REAL    TCWeightY      (MAX_TRACKSP2)
      REAL    TCMinDistance  (MAX_TRACKSP2)
```

```
INTEGER TCNumCor        (MAX_TRACKSP2)
INTEGER TCCenID         (MAX_TRACKSP2)
REAL    TCCenAcoorX     (MAX_TRACKSP2)
REAL    TCCenAcoorY     (MAX_TRACKSP2)
REAL    TCCenIcoorX     (MAX_TRACKSP2)
REAL    TCCenIcoorY     (MAX_TRACKSP2)
INTEGER TCCenArea       (MAX_TRACKSP2)
INTEGER TCCenIntensity  (MAX_TRACKSP2)

INTEGER TFTrackID     (MAX_TRACKSP1)
INTEGER TFNumCor      (MAX_TRACKSP1)
INTEGER TFNumMiss     (MAX_TRACKSP1)
REAL    TFEstimateX   (MAX_TRACKSP1)
REAL    TFEstimateVX  (MAX_TRACKSP1)
REAL    TFXP11        (MAX_TRACKSP1)
REAL    TFXP12        (MAX_TRACKSP1)
REAL    TFXP22        (MAX_TRACKSP1)
REAL    TFEstimateY   (MAX_TRACKSP1)
REAL    TFEstimateVY  (MAX_TRACKSP1)
REAL    TFYP11        (MAX_TRACKSP1)
REAL    TFYP12        (MAX_TRACKSP1)
REAL    TFYP22        (MAX_TRACKSP1)
INTEGER TFIntensity   (MAX_TRACKSP1)
INTEGER TFCSO         (MAX_TRACKSP1)

 COMMON /TC/ TCTrackNum, TCPriority,
+            TCRPriority,
+            TCPositionX,
+            TCPositionY,
+            TCWeightX, TCWeightY,
+            TCMinDistance,
+            TCNumCor, TCCenID,
+            TCCenAcoorX,
+            TCCenAcoorY,
+            TCCenIcoorX,
+            TCCenIcoorY,
+            TCCenArea,
+            TCCenIntensity
 COMMON /TF/ TFTrackID, TFNumCor,
+            TFNumMiss,
+            TFEstimateX,
+            TFEstimateVX,
+            TFXP11, TFXP12,
+            TFXP22,
```

```
     +             TFEstimateY,
     +             TFEstimateVY,
     +             TFYP11, TFYP12,
     +             TFYP22,
     +             TFIntensity, TFCSO
      COMMON /Measure/ MeasurementNoise

      INTEGER I, Ptr, TFPtr

      DO 820 I = 1, (LastTrack - 1)
C       /* Check for multiple centroid associations */
        TFPtr = TCTrackNum (I)
        IF ((TCNumCor (I).GT.0).AND.(TCCenID (I).GE.1)) THEN
          Ptr = I + 1
  810     IF (Ptr.LT.LastTrack) THEN
            IF (TCCenID (I).EQ.TCCenID (Ptr)) THEN
              IF (TCMinDistance (I).GT.TCMinDistance (Ptr)) THEN
                TCCenID (I) = -1
              ELSE
                TCCenID (Ptr) = -1
              ENDIF
            ENDIF
            Ptr = Ptr + 1
            GOTO 810
          ENDIF
        ENDIF

        IF (TCNumCor (I).EQ.0) THEN
C         /* No correlation with this track file */
          IF (TFNumMiss (TFPtr).GE.MAX_MISS) THEN
            TCPriority (I) = 0
          ELSE
            TFNumMiss (TFPtr) = TFNumMiss (TFPtr) + 1
            TCPriority (I) = TFNumCor (TFPtr) - TFNumMiss (TFPtr)
          ENDIF

        ELSEIF (TCCenID (I).EQ.-1) THEN
C         /* Correlation but another track file is closer */
          TFNumMiss (TFPtr) = 0
          TCPriority (I) = TFNumCor (TFPtr)
          CALL KalmanMeasurementUpdate (TCPositionX (I),
     +              TFEstimateX (TFPtr), TFEstimateVX (TFPtr),
     +              TFXP11 (TFPtr), TFXP12 (TFPtr), TFXP22 (TFPtr))
          CALL KalmanMeasurementUpdate (TCPositionY (I),
```

```
     +           TFEstimateY (TFPtr), TFEstimateVY (TFPtr),
     +           TFYP11 (TFPtr), TFYP12 (TFPtr), TFYP22 (TFPtr))
        ELSE
          TFNumCor (TFPtr) = TFNumCor (TFPtr) + 1
          TFNumMiss (TFPtr) = 0
          TCPriority (I) = TFNumCor (TFPtr)
          CALL KalmanMeasurementUpdate (TCCenAcoorX (I),
     +           TFEstimateX (TFPtr), TFEstimateVX (TFPtr),
     +           TFXP11 (TFPtr), TFXP12 (TFPtr), TFXP22 (TFPtr))
          CALL KalmanMeasurementUpdate (TCCenAcoorY (I),
     +           TFEstimateY (TFPtr), TFEstimateVY (TFPtr),
     +           TFYP11 (TFPtr), TFYP12 (TFPtr), TFYP22 (TFPtr))
        ENDIF
  820 CONTINUE

      RETURN
      END
```

4. References

[BDM1] BDM Corporation, " Exo-Atmospheric Staring Seeker/Image Emulator Generator *(ESSIEG) Emulator 1 and Emulator 2 Considerations*" Contract No.: DASG60-85-C-0041. Prepared for Georgia Institute of Technology

5. Appendices

Appendix A: AEDC Clustering Test Program

Program listing withheld for competitive reasons.